
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**
**DEPARTMENT OF COMPUTER SYSTEMS**


# NOVÁ GENERACE IPFIX KOLEKTORU

**A NEW GENERATION OF AN IPFIX COLLECTOR**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** — **Bc. LUKÁŠ HUTÁK**
**AUTHOR**

**VEDOUCÍ PRÁCE** — **Ing. JAN WRONA**
**SUPERVISOR**



**BRNO 2018**

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačových systémů Akademický rok 2017/2018

# Zadání diplomové práce

Řešitel: **Huták Lukáš, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Nová generace IPFIX kolektoru**
**A New Generation of an IPFIX Collector**
Kategorie: Počítačové sítě

Pokyny:

1. Nastudujte protokol IPFIX a jeho využití při monitorování síťového provozu.
2. Seznamte se s modulárním kolektorem IPFIXcol, který je vyvíjen organizací CESNET.
3. Analyzujte současný stav a navrhněte architekturu a softwarové rozhraní nové generace kolektoru.
4. Implementujte nové jádro kolektoru a přizpůsobte vybrané existující pluginy novému rozhraní.
5. Diskutujte dosažené výsledky a možná rozšíření.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

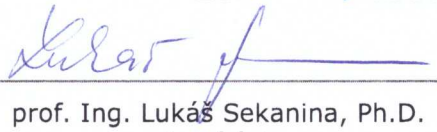
Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Wrona Jan, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2



prof. Ing. Lukáš Sekanina, Ph.D.
*vedoucí ústavu*

# Abstrakt

Tato práce se zabývá zpracováním záznamů toků vzniklých monitorováním provozu počítačových sítí z pohledu IPFIX kolektoru. Analyzuje současné řešení modulárního kolektoru, který již prošel značným historickým vývojem, a zaměřuje se na odhalení jeho silných a slabých stránek. Na základě získaných znalostí navrhuje nový kolektor, který významně upravuje řešení jednotlivých komponent pro práci se záznamy toků, klade důraz na vysokou propustnost a přidává dosud chybějící funkcionality. V rámci práce došlo i ke srovnání výkonnosti obou generací, které vyznívá pozitivně pro nový kolektor.



# Abstract

This master's thesis addresses processing of flow monitoring records from a point of view of an IPFIX collector. It analysis the current solution of the modular collector, which went through considerable historical development, and focuses on revealing its strengths and weaknesses. Based on acquired knowledge, a new collector is designed. The new solution, which significantly modifies individual components for processing of flow records, focuses on high throughput and adds missing functionalities. The document also compares performance of both generations and the new collector clearly dominates.



# Klíčová slova

Monitorování sítí, sběr dat, IPFIX, NetFlow, kolektor, tok, paralelizace, bezpečnost, počítačové sítě, nová archtektura, zřetězená linka, modularita

# Keywords

Network monitoring, data collection, IPFIX, NetFlow, collector, flow, parallelization, computer networks, new architecture, pipeline, modularity


# Citace

HUTÁK, Lukáš. *Nová generace IPFIX kolektoru*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Wrona

# Nová generace IPFIX kolektoru

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Wrony. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Huták
21. května 2018

## Poděkování

Rád bych poděkoval organizaci CESNET z.s.p.o. a zejména členům jejího výzkumného týmu Liberouter za podporu, bez které by tato práce nemohla vzniknout. Rád bych též poděkoval své rodině, která mi byla vždy oporou po dobu mého studia.

# Obsah

**1 Úvod** **3**

**2 Sítě a monitorování** **5**
2.1 Metody sledování sítě . . . . . 5
2.2 NetFlow a IPFIX . . . . . 6
2.3 Zachycení provozu . . . . . 8
2.4 Měření, expirace a export záznamů o tocích . . . . . 9
2.5 Sběr záznamů o tocích a analýza . . . . . 12

**3 Protokol IPFIX a jeho struktura** **15**
3.1 Historie a vývoj protokolu . . . . . 15
3.2 Informační elementy a datové typy . . . . . 16
3.3 Datové struktury použité v protokolu . . . . . 18
3.4 Správa šablon a vliv transportních protokolů . . . . . 20
3.5 Vybraná rozšíření protokolu . . . . . 23

**4 Kolektor současné generace a jeho analýza** **25**
4.1 Architektura kolektoru a proud dat . . . . . 25
4.2 Preprocesor IPFIX zpráv a správa šablon . . . . . 27
4.3 Rozhraní zásuvných modulů a jejich zřetězení . . . . . 28
4.4 Způsob práce se záznamy . . . . . 29
4.5 Konfigurace kolektoru a správa informačních elementů . . . . . 30
4.6 Stav kódu, testovatelnost a dokumentace . . . . . 31

**5 Kolektor nové generace** **33**
5.1 Architektura jádra kolektoru . . . . . 33
5.2 Informační elementy a jejich správa . . . . . 39
5.3 Šablony a jejich správa . . . . . 42
5.4 Zpracování zpráv a práce s datovými záznamy . . . . . 49

**6 Realizace kolektoru a měření dosažených výsledků** **53**
6.1 Struktura projektu . . . . . 53
6.2 Jádro kolektoru . . . . . 55
6.3 Zásuvné moduly . . . . . 58
6.4 Budoucí rozvoj projektu . . . . . 65

**7 Závěr** **67**

**Literatura** **69**

**A Obsah přiloženého paměťového média** **71**

**B Překlad a spuštění kolektoru** **72**

**C Konfigurace nového kolektoru** **73**

**D Škálovatelnost instancí vstupních modulů** **76**

# Kapitola 1

# Úvod

Monitorování sítí představuje zásadní bezpečnostní prvek pro jejich provoz a správu. Jednou z významných oblastí je monitorování síťového provozu a s tím související technologie označované jako NetFlow a IPFIX. Hlavní myšlenka spočívá v analýze procházejících paketů na vybraných bodech sítě, extrakci informací o síťových tocích a jejich průběžném exportování na sběrné místo označované jako kolektor. Kolektor jakožto kriticky důležitý centrální bod obvykle příchozí data ukládá pro možnou manuální analýzu. Vzhledem k tomu, že se v dnešní době obecně přikláníme k proaktivnímu přístupu, už pouhé ukládání nestačí a kolektor by tak měl být schopen poskytovat data systémům pro automatizovanou analýzu provozu. Významnou funkčností kolektoru se tudíž stává i modularita, která dovoluje značně rozšířit jeho působnost a univerzálnost. Zároveň nelze opomenou důraz na vysokou výkonnost vzhledem k potřebám nasazení pro sběru dat z různých míst páteřní sítě.

Kolektor IPFIXcol, vyvíjený při organizaci CESNET z.s.p.o., splňuje výše uvedené předpoklady na modularitu i propustnost. Podporuje sadu vstupních, vnitřních a výstupních modulů, přičemž umožňuje uživatelům si vybrat vhodnou kombinaci příjmu dat, jejich úpravy a výstupu. Avšak samotný nástroj prošel již značným historickým vývojem, který se na něm v řadě ohledů podepsal. Jako v případě mnohých softwarových produktů postupným přidáváním zpočátku nepředpokládaných funkcionalit, ba dokonce zanedbáním některých elementárních postupů se stal samotný následující vývoj neudržitelný a těžkopádný. Noví uživatelé a přispěvatelé se v nástroji poměrně snadno ztrácejí, dopouštějí se chyb a řadu nových funkcionalit již není možné vhodně implementovat. Náplní této práce je tudíž mimo jiné podrobně analyzovat současný stav kolektoru, využít dosavadních znalostí nabytých během přispívání do projektu k identifikaci silných a slabých míst současného návrhu a navrhnout vylepšenou architekturu a softwarové rozhraní, které bude základem nové generace tohoto kolektoru. Nový návrh se tak bude zabývat nejen řešením současných palčivých problémů, ale z hlediska dalšího směřování a rozvoje musí nezbytně odhadovat a zároveň opatrně vytyčovat i směr budoucího vývoje, aby nedošlo k opětovnému výskytu stěží implementovatelných funkčností.

Text diplomové této práce je strukturován dále do samostatných logických celků. Navazující 2. kapitola detailněji proniká do metody měření síťových toků pomocí technologií NetFlow a IPFIX, komponent pro ni nezbytných a použitých principů. Pochopení klíčových částí kolektoru a způsobu práce s přenášenými daty si vyžaduje základní znalost vlastností a rozsahu schopností protokolu IPFIX, a proto je mu věnována celá kapitola 3. Před samotným návrhem nové architektury kolektoru je v kapitole 4 rozebrána podoba současného kolektoru, kde se detailně pojednává o jednotlivých jeho komponentách se všemi přednostmi i stinnými stránkami. Dále navazující 5. kapitola pak ze získaných znalostí vychází, diskuje

problémy dosavadních řešení a nabízí nové přístupy. Implementaci a zejména testování nově vzniklého kolektoru je věnována kapitola 6. V jejím rámci dochází mimo jiné k výkonostnímu porovnání staré a nově vzniklé generace kolektoru. Závěrečná 7. kapitola shrnuje dosažené výsledky této práce.

# Kapitola 2

# Sítě a monitorování

S každým rokem množství provozu na počítačových sítích stoupá a nic nenasvědčuje tomu, že by se tento rostoucí trend měl výhledově měnit. Jejich správci tak nemají na výběr a musí dohlížet na aktuální stav a odhadovat budoucí vývoj. Analogicky bychom to mohli přirovnat k monitorování provozu na automobilových dálnicích a městských křižovatkách. Bez přítomnosti dohledového systému jsme zcela slepí a naše schopnosti jsou omezeny na pouhé reakce na vzniklé události, což je z pohledu plynulosti a všeobecné spokojenosti účastníků zásadně pozdě. Je-li znám aktuální stav a očekávaný výhled, mohou se tyto získané znalosti využít pro lepší plánování, snazší řešení vzniklých událostí i pro proaktivní přístup v jejich předcházení. To stejné platí jak v dopravě, tak i počítačových sítích. Opomenout monitorování sítí může mít v samotném důsledku fatální následky.

Provoz každé větší počítačové sítě obvykle vyžaduje prostředky pro sledování jejího stavu. Přístupů a technologií existuje celá řada. Pokrývají oblasti od sledování stavu zařízení podílejících se na přenosu dat skrze síť nebo zařízení k síti připojených až po sledování samotného obsahu přes síť plynoucího provozu. Náplň této kapitoly se zaměřuje na základní přehled posledně jmenované oblasti, a to z pohledu monitorování síťových toků, jejich nezbytných komponent a využívaných principů.

## 2.1 Metody sledování sítě

Než bude rozebrána samotná metoda sledování toků, je vhodné si ji stručně zařadit do kontextu ostatních technologií používaných pro monitorování a správu sítě. Monitorování můžeme rozdělit na aktivní nebo pasivní [10].

V případě **aktivního monitorování** sítě obvykle dochází k periodickému dotazování na sledované zařízení a interpretaci jeho případných odpovědí. Jinými slovy, při monitorování se aktivně zahajuje komunikace nebo generuje testovací provoz na sledované zařízení, které na požadavky přímo reaguje. Pokud zařízení opakovaně neodpovídá nebo získané údaje nesplňují požadovaná kritéria, je správce sítě na tuto skutečnost automaticky upozorněn. Mezi poměrně známé zástupce této kategorie lze zařadit generování ICMP zpráv s dotazy na dosažitelnost cílového zařízení nebo použití protokolu SNMP pro získání parametrů a stavů běžícího zařízení. Při aktivním monitorování se nesmí opomíjet na fyzické uspořádání sítě a s tím související závislost jedné služby na službách jiných. Dosti ilustrativním případem je situace výpadku směrovače (nebo přepínače), za kterým se nachází servery poskytující jednu nebo více sledovaných služeb. Jeden výpadek tak má za následek masivní příval hlášení

nedostupnosti celé řady služeb. Znalost hierarchie závislostí je tak nutná pro pochopení hlášení a identifikace skutečného bodu poruchy.

**Pasivní monitorování** se vyznačuje zejména sledováním probíhajícího provozu na síti. Analýzou uživatelského provozu jsme schopni zjistit užitečná data o využití sítě bez toho, aniž by uživatel měl šanci se dozvědět, že jeho provoz je analyzován a zároveň účelně nezvyšujeme provoz na síti. Použití má širokou škálu využití v oblastech jako bezpečnost, správa a plánování rozvoje sítě nebo účtování poskytnutých služeb. Do kategorie pasivního monitorování lze zařadit i autonomní hlášení ze sledovaných zařízení na sběrný bod, ať už v podobě pravidelných statistik, nebo hlášení zaznamenaných událostí, neboť původcem komunikace je sám sledovaný.

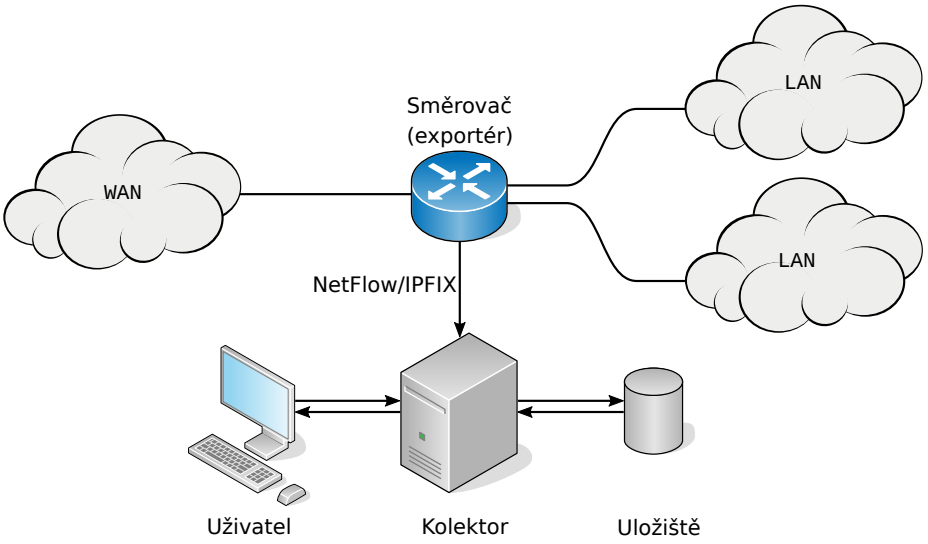
Z uvedeného plyne, že je třeba odlišovat sledování stavu jednotlivých zařízení (např. přepínačů, směrovačů, serverů a jiných) v síti od monitorování samotného procházejícího provozu přes síť. Zatímco při zjišťování stavu zařízení se ověřuje, zda jsou v provozu schopném stavu a jejich provozní parametry nevybočují od normálu, nejsou tato zařízení obvykle sama schopna poskytnou podrobné informace o zpracovávaném provozu. V tu chvíli nastupují technologie monitorování síťového provozu, které analyzují obsah komunikace ve vybraných místech sítě, ale na druhou stranu neumožňují přímočaře získat podrobnosti o vnitřním stavu zařízení v síti. Z výše uvedených kategorií lze zařadit monitorování síťových toků do kategorie pasivního monitorování zaměřeného na sledování provozu.

## 2.2 NetFlow a IPFIX

Kolem monitorování toků vzniklo hned několik technologií, přičemž asi nejznámější je NetFlow od firmy Cisco. Pod daným označením se obvykle skrývá soubor zařízení schopných měření toků a jejich zpracování, ale i samotný komunikační protokol pro přenos naměřených toků. Historie technologie sahá až do roku 1996, kde však původní ideou bylo její využití pro urychlení směrování paketů na směrovačích, kdy prvním paketem toku byl založen záznam o směrování. Tento záznam po dobu své životnosti sloužil k určení, kam mají další pakety náležící stejnému toku být směrovány [1]. Navíc se u jednotlivých záznamů uchovávaly informace související se samotným tokem (celkové množství paketů, bytů apod.), což se ukázalo jako poměrně hodnotné pro představu o provozu na síti. Postupem času se tudíž účel technologie transformoval do dnešní podoby a ta se stala běžnou na poli monitorování toků. Standard IPFIX představuje nástupce, který si klade za cíl rozšířit a sjednotit technologii mezi všemi výrobci, kteří si původní NetFlow přizpůsobovali svým potřebám.

Na rozdíl od postupů, kdy se uchovává veškerý provoz sítě na úrovni paketů (angl. *packet capture*), se NetFlow a IPFIX snaží získat pouze statistiky jednotlivých komunikací uživatelů za delší časové období. Analogicky lze množství získávaných informací přirovnat k výpisu telefonních hovoru, kde lze zjistit kdo, s kým, kdy a jak dlouho komunikoval, ale samotný obsah komunikace uchován není. Tím, že se analyzují a uchovávají jen vybrané vlastnosti komunikace je možné monitorovat i vysokorychlostní páteřní sítě a výsledky dlouhodobě skladovat, což v případě uchování veškerých paketů nepřipadá v úvahu, a to zejména kvůli enormním nárokům na kapacitu a propustnost uložiště.

Jako alternativní technologie se poměrně často označuje sFlow, která ovšem s NetFlow a IPFIX má pramálo společného. Na rozdíl od výše zmíněných nesleduje jednotlivé toky, ale pouze vzorkováním náhodně vybírá 1 z N zachycených paketů a je jich hlavičky nebo z nich extrahované parametry posílá na sběrné místo. Navíc dokáže sledovat celkové statistiky provozu na jednotlivých rozhraních. Lze tedy vidět, že slovo „*flow*“ představuje více méně




Obrázek 2.1: Typická architektura monitorování toků

jen marketingové označení. Vzhledem k tomu, že tato práce se zaobírá IPFIX kolektorem, navazující text této kapitoly je pojednán z pohledu NetFlow a IPFIX.

Základním pojmem, který je a bude často skloňován, je **tok** (angl. *flow*). Jak připouští standard RFC 7011 [4], používaných definic v rámci internetové komunity existuje hned několik, ovšem základ většiny definic je přitom velmi podobný. Níže je uvedena definice používaná v rámci této práce:

*Síťový tok je posloupnost paketů procházející bodem pozorování během určitého časového intervalu, kde všechny pakety náležící stejnému toku mají množinu společných vlastností.*

Mezi vlastnosti sledované u toků se typicky řadí položky získatelné nebo odvozené ze síťových a transportních hlaviček paketů. Meze se ničemu nekladou, a tak mohou být např. sledovány i položky z linkové a aplikační vrstvy. Vlastnosti se z pohledu přítomnosti v paketech mohou rozdělit na společné, které jsou součástí každého paketu (např. zdrojová IP adresa a port), a vlastnosti přítomné jen v některých paketech, vypočítané či jinak odvozené (např. celkový počet přenesených bytů). Podmnožina společných vlastností jejichž kombinace hodnot se používaná k určení, ke kterému toku paket náleží, se označuje jako *klíč toku* (angl. *flow key*) a může se lišit napříč zařízeními provádějící měření toků. Obvyklým základem ovšem bývá 5tice vlastností: zdrojová a cílová IP adresa, zdrojový a cílový port a protokol.

Tok, jak je zde definován i vzhledem ke klíčovým vlastnostem, představuje záznam o jednosměrném proudu dat. Obvyklá komunikace mezi klientem a serverem ovšem bývá obousměrná, což v důsledku při měření směřuje k vytvoření dvou záznamů o tocích – od klienta k serveru a od serveru ke klientovi – jak je vidět na obrázku 2.2. Alternativou je **obousměrný tok** označovaný jako *Biflow (Bidirectional Flow)* [17], jenž v sobě kombinuje oba zmíněné samostatné toky, a tak umožňuje zaznamenat komunikaci v souvislostech.

Základní komponenty, z nichž se typická architektura skládá, jsou exportér, kolektor a komunikační protokol pro přenos naměřených toků. Z obrázku 2.1 je patrné, že exportér monitoruje provoz na vybraném bodě sítě a provádí měření toků analyzováním zachycených paketů. Jako exportér může vystupovat běžný síťový prvek obohacený o schopnost měření toků (např. směrovač), speciální síťová sonda nebo počítač vybavený síťovou

kartou a odpovídajícím softwarem. Naměřené statistiky o tocích se po určité době exportují na kolektor vygenerováním zpráv dle protokolu NetFlow nebo IPFIX. Kolektor přijímá zprávy obvykle od více exportérů, data analyzuje a záznamy ukládá na dlouhodobé uložiště či poskytuje dalším aplikacím pro analýzu. Pro správce jsou často dostupné nástroje tvořící přehledy a vizualizace nad uloženými daty, které umožňují realizovat i manuální rozbor. Celou činnost lze rozdělit do několika samostatných celků od zachycení provozu až po analýzu toků, jímž se věnují následující sekce, jejichž struktura vychází z poznatků uvedených ve článku Flow Monitoring Explained [6]. Za zmínku dále stojí fakt, že standard IPFIX [4] mírně upravil definici exportéru, kdy u něj nově odděluje činnosti na tzv. *měřicí proces* (angl. *Metering Process*), který provádí činnosti od zachycení provozu až po tvorbu záznamů o tocích, a *exportní proces* (angl. *Exporting Process*), jehož náplní je čistě přenos dat ke kolektoru.

## 2.3 Zachycení provozu

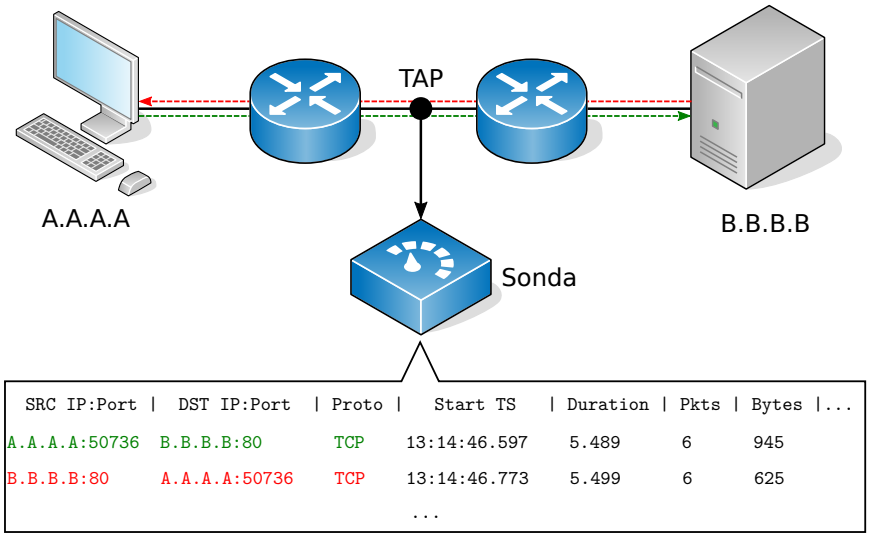
Celý proces měření toků na exportéru začíná odchycením datového provozu z komunikační linky na vybraném místě sítě označeném jako *pozorovací bod* (angl. *Observation Point*). Zachycení provozu může probíhat přímo tak, že zařízení je vloženo mezi dva body v síti a provoz skrze něj prochází (např. směrovač, viz. obrázek 2.1) nebo kopie komunikace je na něj odbočena a o takových zařízeních se obvykle mluví jako o sondách. Pakety se na síťových kartách monitorovacích zařízení zachytí a neprovádí-li se dále hloubková analýza provozu, může být paket zkrácen jen na nezbytně nutné hlavičky, neboť ty obvykle představují hlavní zdroj pro měření toků. Fragment paketu by si tak mohl zachovat pouze hlavičku vrstvy síťového rozhraní (typicky Ethernet), hlavičku síťové vrstvy (IPv4/IPv6) a hlavičku vrstvy transportní (TCP/UDP). Zkrácení má obvykle příznivý vliv na snížení náročnosti zpracování provozu.

Množství provozu na zpracování je možné ještě více zredukovat pomocí filtrování a vzorkování. Vzorkováním se systematicky nebo náhodně vybírá 1 z N paketů, který se posílá na další zpracování a zbytek je zahozen. Dopad takového přístupu je zásadní, jelikož informace o některý tocích mohou buď zcela chybět nebo mohou být značně zkreslené, což snižuje celkovou informační hodnotu a eliminuje uplatnitelnost např. v oblasti detekce provozních anomálií nebo účtování provozu. Filtrování provozu je na tom podstatně lépe. Opět redukuje množství paketů, avšak při vhodném nastavení filtračních podmínek umožňuje vybrat provoz tak, že pakety zájmových toků budou bez povšimnutí dále propuštěny a jiné budou zahozeny. Některé toky tak sice nejsou zachyceny, ale u toků vybraného provozu nedojde ke zkreslujícím výsledkům. Filtry se zaměřují zejména na relativně snadno analyzovatelné vlastnosti z hlaviček jako jsou např. IP adresy, porty a protokoly. Jelikož vzorkování a filtrování může být akcelerované přímo v hardwaru síťových karet, k softwarovému zpracování se posílá pouze fragment dat a nároky na procesor jsou o poznání nižší.

Pokud záchyt probíhá nad počítačem se síťovou kartou, obvykle je nutné dostat takto získaná data do aplikace provádějící následnou analýzu paketů a měření toků. V tomto ohledu lze využít rozšířené a všeobecně podporované knihovny *libpcap* (Linux) a *winpcap* (Windows) postavené nad universálním systémovým síťovým rozhraním. Stále častěji se začínají používat knihovny jako *DPDK*[1] a *PF_RING*[2], které obchází systémové rozhraní

[1] http://dpdk.org/

[2] http://www.ntop.org/products/packet-capture/pf_ring/




Obrázek 2.2: Příklad monitorování síťových toků pomocí sondy s ukázkou obsahu záznamů v mezipaměti toků

a zprostředkovávají přímo komunikaci mezi kartou a aplikací bez režie zavedené systémem. U tohoto řešení je omezující menší množství podporovaných síťových karet.

Vcelku zajímavý směr představuje i řešení [7], kdy méně zajímavá část toků je celá zpracována v samotném hardwaru speciální síťové karty a pakety toků, u nichž se provádí hloubková analýza obsahu, jsou odeslány na zpracování do softwaru.

## 2.4 Měření, expirace a export záznamů o tocích

Na prvotní fázi získání a předzpracování paketů navazuje fáze transformace paketů do podoby toků. Tato činnost zahrnuje identifikaci náležitosti paketu do toku, jeho agregaci do společného záznamu a spravování záznamů toků.

Měření toků je dlouhodobá záležitost. Vlastnosti toků se udržují v podobě záznamů po dobu své existence v *mezipaměti toků* (angl. *flow cache*). První činností u každého přijatého paketu je proces určení, ke kterému toku náleží. Jak již bylo zmíněno u popisuje definice toku v podkapitole 2.2, některé sledované vlastnosti jsou označeny za klíčové a kombinace jejich hodnot je unikátní pro každý tok. Extrakcí těchto vlastností se získá unikátní identifikátor v mezipaměti pro nalezení záznamu, se kterým se dále pracuje.

Základní podoba záznamu toků s drobnými variacemi společná napříč implementacemi měřících procesů různých výrobců je uveden v tabulce 2.1. Součástí záznamu jsou již zmiňované klíčové vlastnosti (v tabulce označeny symbolem "▷") společně s neklíčovými a jelikož klíčové vlastnosti jsou stejné pro všechny pakety toku, po celou dobu existence záznamu zůstávají neměnné. S příchodem každého paketu toku nastává aktualizace neklíčových vlastností záznamu, a to agregací hodnot z nově přijatých paketů. Agregační metoda se liší od vlastnosti k vlastnosti např. u celkového počtu bytů se hodnota velikosti přijatého paketu přičítá k průběžné hodnotě, u počtu paketů se hodnota čítače inkrementuje apod.

Výčet sledovatelných vlastností není ani zdaleka úplný a každý tvůrce měřícího procesu může obohatit toky o další vlastnosti. Zajímavou přidanou hodnotu poskytují informace získatelné z aplikačních protokolů. Příkladem může být extrakce z webového protokolu HTTP, kdy lze z paketů získat informace o navštívené webové stránce a prohlížeči, nebo

| | Sledované vlastnosti | Datový typ | Inf. elementy IPFIX[3] |
|---|---|---|---|
| ▷ | Zdrojová IP adresa | IPv4/IPv6 adresa | sourceIPv4Address<br>sourceIPv6Address |
| ▷ | Cílová IP adresa | IPv4/IPv6 adresa | destinationIPv4Address<br>destinationIPv6Address |
| ▷ | Zdrojový port | Nezáporné číslo | sourceTransportPort |
| ▷ | Cílový port | Nezáporné číslo | destinationTransportPort |
| ▷ | Protokol | Nezáporné číslo | protocolIdentifier |
| | Čas prvního paketu | Datum | flowStartMilliseconds |
| | Čas posledního paketu | Datum | flowEndMilliseconds |
| | TCP příznaky | Nezáporné číslo | tcpControlBits |
| | Počet bytů | Nezáporné číslo | octetDeltaCount |
| | Počet paketů | Nezáporné číslo | packetDeltaCount |

Tabulka 2.1: Položky tvořící typický základ záznamů toků

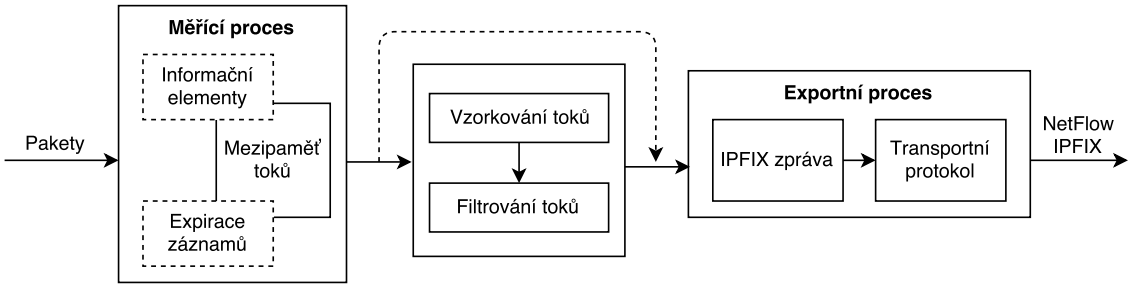
emailového protokol SMTP, z něhož je možné získání adresy odesílatele, adresáta i obsah samotné zprávy. Byť takto nabyté informace poskytují cenné poznatky pro detekci nestandardně chovajících se uživatelů, míra analýzy už začíná citelně zasahovat do soukromého obsahu a zároveň signifikantně zvyšuje nároky na potřebném výpočetní prostředky. S rostoucím popularitou šifrovaného přenosu se k aplikačnímu obsahu už není tak jednoduché dostat jako dříve a je otázkou, zda tento směr neztratí zcela význam.

Množství záznamů v mezipaměti toků se s plynoucím časem postupně zvyšuje, aby ale nedošlo k jejímu zaplnění a tím pádem k nemožnosti vytvářet záznamy nové, jsou průběžně toky označovány za ukončené, odebrány z mezipaměti a exportovány. Pokud později přijde další paket náležící odebranému toku, vytvoří se nový záznam a tvorba probíhá nanovo. Toky mohou být označeny za ukončené kvůli jednomu z níže uvedených důvodů:

**Aktivní časový limit** Maximální stanovená doba existence pro příliš dlouho trvající pravidelně aktualizované záznamy toků, kdy po jejím uplynutí je záznam násilně ukončen, byť komunikace stále probíhá. Pokud by tento limit hypoteticky neexistoval, dlouhotrvající toky (v řádu hodin i dnů) by nebyly až do svého ukončení nikdy odeslány na kolektor a správce by se tak neměl možnost průběžně dovědět o této činnosti na síti nebo by informace získal se značným zpožděním. Na druhou stranu důsledkem uplatňování tohoto limitu je, že na straně kolektoru bude jeden dlouhotrvající tok fragmentován do podoby řady menších na sebe navazujících toků. Obvyklá hodnota limitu je v řádu jednotek minut.

**Neaktivní časový limit** Záznamy, u nichž nedošlo k žádné aktualizaci po stanovenou dobu, jsou násilně ukončeny. Má značný význam u toků nad transportními protokoly (např. UDP), u nichž nelze jednoduše rozeznat ukončení komunikace, nebo v případě, kdy jedna strana nenadále přestala komunikovat. Limit nabývá hodnot obvykle v řádu desítek sekund.

[3]Uvedené identifikátory představují příklady názvů informačních elementů používaných v rámci protokolu IPFIX pro označení odpovídajících vlastností




Obrázek 2.3: Architektura fáze měření a exportu toků [6]

**Detekované ukončení toku** V případech, kdy je možné rozeznat ukončení komunikace (typicky TCP paket s příznakem FIN nebo RST) je přímo vhodné záznam toku ukončit. Pokud by komunikace se stejnými klíčovými vlastnostmi byla opětovně zahájena, nedojde tak ke smíchání vlastností z komunikace nové a staré, ale jsou korektně vytvořeny záznamy zcela separátní.

**Jiné** Zaplnění mezipaměti nad takovou mez, že již není možné bezproblémově vytvářet záznamy nových toků, nebo ukončení zbývajících toků v okamžiku ukončení činnosti měření, vynucené ukončení vybraných drůhů toků na základě jiných vnitřních podmínek pro jejich zpracování apod.

Nejvýznamnější vliv na zaplněnost tabulky a množství exportovaných toků, pokud pomineme samotnou kapacitu mezipamětí v kontextu odhadované vytížeností monitorované sítě, má nastavení časových limitů. Příliš nízký neaktivní časový limit může v některých případech způsobit, že záznamy toků budou předčasně exportovány, což na straně kolektoru zvýší množství přijatých toků. Naopak pokud aktivní časový limit bude příliš vysoký, množství exportovaných toků se sice sníží, ale záznamy toků jsou pro analýzu k dispozici s větším zpožděním.

Před odesláním na kolektor může ještě volitelně (dle konfigurace) dojít ke vzorkování a filtrování podobně jak tomu bylo při záchytu paketů. Jakmile jsou záznamy vybrány pro odeslání, jsou předány exportnímu procesu, který přebírá zodpovědnost za přenos dat na jeden nebo více kolektorů. Pro komunikaci s exportérem se používá jednosměrný protokol NetFlow v5/v9 nebo nověji protokol IPFIX. Samotná struktura protokolu IPFIX, jeho schopnosti a vlastnosti jsou detailně rozebírány v samostatné 3. kapitole. Předem lze však prozradit, že záznamy o tocích musí být exportním procesem speciálně naformátovány a zdaleka nemusí mít všechny stejnou podobu, neboť jak bylo dříve uvedeno, některé záznamy toků mohou být rozšířeny o aplikační data nebo jiné pro ně specifické vlastnosti. Pokud je to v závislosti na vybraném protokolu a jeho verzi možné, exportní proces strukturu jednotlivých typů záznamů zadefinuje pomocí mechanismu šablon.

Naformátované NetFlow či IPFIX zprávy jsou následně přenášeny na jeden nebo více kolektorů za pomocí transportního protokolu TCP, UDP případně SCTP. Povaha obsahuje zpráv je značně citlivého charakteru a jejich únik do rukou potenciálního útočníka při přenosu mezi exportérem a kolektorem by mu mohl pomocí zmapovat síť nebo získat informace o návycích uživatelů. Nabízí se tak případně možnost přenášené zprávy navíc šifrovat za pomocí ochrany TLS (Transport Layer Security) pro TCP či DTLS (Datagram Transport Layer Security) pro UDP a SCTP. Sama volba transportního protokolu má významné důsledky na doručení zprávy.

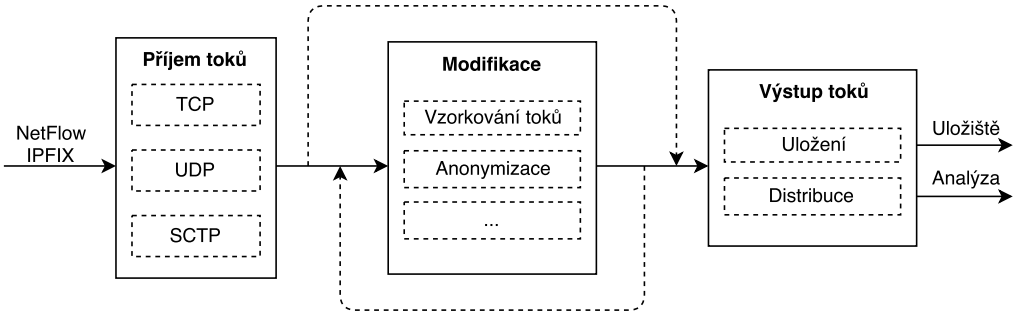
Z uvedené trojice nejjednodušší protokol **UDP** (User Datagram Protocol) je kvůli své snadné implementovatelnosti a nenáročnosti populární nejen při exportu z jednodušších síťových prvků. Jeho použití však sebou nese značná rizika v podobě ztráty některých zpráv nebo přeskládání pořadí na straně příjemce. Protokol IPFIX se s tím umí vypořádat a ve většině případů to vede pouze ke ztrátě záznamů z nedoručených zpráv. Jsou ovšem i situace, kdy ztráta jedné zprávy může mít vliv na interpretovatelnost záznamů v dalších zprávách. Druhý a neméně známý transportní protokol **TCP** (Transmission Control Protocol) se, co se týče vlastností, jeví jako přesný opak. Potvrzování přijetí zpráv příjemcem garantuje, že i v případě ztráty jakékoliv zprávy při přenosu bude zaslána její nová kopie. To v kombinaci s garancí pořadí doručení umí zaručit spolehlivý přenos zpráv. Avšak tím, že zároveň kontroluje zahlcení komunikačního kanálu, pokud vzniká problém při přenosu dat nebo pokud příjemce dostatečně rychle neodebírá zprávy, může způsobovat paradoxně problémy exportnímu procesu. Ten je blokován a pro udržení svoji korektní činnost je nucen část záznamů kontrolovaně zahazovat. Poslední jmenovaný protokol **SCTP** (Stream Control Transmission Protocol) se řadí mezi novější, nepříliš známé a méně rozšířené protokoly. Oproti uvedeným má zcela unikátní vlastnosti, kdy mezi komunikujícími zařízeními se, pokud je to možné, průběžně vytváří více spojení skrze jejich různá síťová rozhraní, což v případě výpadku nebo zhoršení kvality hlavního z nich, dovoluje přejít na záložní a v komunikaci pokračovat bez rozpadu spojení. Navíc skrze vzájemné spojení je možné vytvořit samostatné proudy dat a ztráta zprávy v jednom z proudů neblokuje při přeposílání proudy jiné. To je zásadní rozdíl oproti protokolu TCP, který ve své podstatě představuje jen jediný proud dat. Mnoho funkcionalit je oproti dříve uvedeným transportním protokolům také konfigurovatelných. Je tak možné vybrat, zda bude zajištěna spolehlivá komunikace nebo se zprávy mohou ztrácet. Případně zda je garantováno doručení v pořadí nebo ne.

## 2.5 Sběr záznamů o tocích a analýza

Data od exportních procesů jsou odeslána na kolektor, jehož náplní je realizovat centralizovaný sběr záznamů. Zdánlivě by se na první pohled mohlo zdát, že veškerá komplikovaná činnost již proběhla během samotného měření toků a sběr představuje už pouhou banalitu. Ukázat, že tomu tak opravdu není, se bude snažit nejen tato sekce ale i zbytek této diplomové práce. Činnosti, do kterých kolektor zasahuje, lze rozdělit do oblastí od příjmu, modifikace a uchování až k analýze záznamů.

Vše začíná **fází přijmu dat** posílaných obvykle z více než jednoho exportního procesu. Komplikaci představuje nejen to, že odlišné exportní procesy mohou být nakonfigurovány tak, aby používaly odlišný transportní protokol pro přenos dat ke kolektoru, ale také fakt, že mohou použít mírně odlišný aplikační protokol, NetFlow v5/v9 nebo IPFIX. Kolektor, pokud požadovaný transportní či aplikační protokol podporuje, by měl být schopen se přizpůsobit různým kombinacím. Aby situace byla ještě o něco více složitější, každý exportní proces, který definuje strukturu přenášených záznamů toků za pomoci šablon, tak činní zcela nezávisle na ostatních exportních procesech. Pro kolektor to znamená povinnost předpokládat, že šablony (a tím pádem i struktura záznamů) od každého exportního procesu jsou zcela jiné, z čehož v důsledku plyne, že při interpretaci záznamů toků se musí řídit definicemi od konkrétního odesilatele záznamů. Záznamy toků obsahují celou řadu vlastností a může se stát, že určitá podmnožina není kolektorem podporována. V takovém případě obvykle dojde k jejich nevratné ztrátě.

Na příjem paketů navazuje volitelná **fáze modifikace záznamů**. Možností je vskutku nespočet a jsou závislé vždy na konkrétních požadavcích správce a schopnostech kolektoru.




Obrázek 2.4: Architektura sběru a dalšího zpracování toků

Je tedy možné část vlastností každého záznamu odstranit, upravit nebo přidat. Úprava a odstranění se často provádí o okamžiku, kdy požadované vlastnosti jsou např. nadbytečné, chybné nebo je vhodné provést jejich anonymizaci zamezující identifikovat účastníky komunikace. Naopak přidat lze informace např. o tom, kdy tok přišel na kolektor, jaká je odhadovaná geografická poloha komunikujících stran apod. Navíc stejně jako exportní proces i kolektor může provádět filtrování a vzorkování toků. Zatímco na kolektoru tato činnost probíhá nad daty od všech odesílatelů a konfigurace je tak jednotná, v případě exportních procesů, kterých je obvykle více a jsou na sobě nezávislé, musí správce provést konfiguraci na každém zvlášť.

Finální fáze, do níž záznamy vstupují, se zabývá **uložením a distribucí toků** dalším nástrojům. Perzistentní uchování záznamů je asi nejběžnější činností pro případ pozdější manuální analýzy. Přístupů k uchování je celá řada a de facto stejná problematika je sdílena napříč celým odvětvím informačních technologií. Záznamy mohou být uchovány v podobě textové nebo binární. V prvním z případu jsou uživatelsky – více či méně – čitelné i bez použití přídavných nástrojů, snadno přenositelné a upravitelné. Avšak diskový prostor, který ke svému uchování vyžadují, je znatelně vyšší, dochází ke zbytečnému plýtvání místa a pokročilejší práce v podobě filtrování a dotazování nad daty vyžaduje použití specializovaných nástrojů, které stejně textovou informaci musí binárně interpretovat. Naopak binární reprezentace je značně kompaktnější i rychlejší pro dotazování, ale vždy vyžaduje odpovídající program schopný s daty pracovat.

V oblasti ukládání toků je poměrně rozšířený přístup používaný např. v nástrojích postavených okolo programu *nfdump*[4]. Jeho jednoduchost spočívá v sekvenčním ukládání celých záznamů do specializovaných souborů v pořadí, v jakém záznamy při sběru přišly. Aby nevznikl jeden rozsáhlý soubor, dochází k pravidelnému obměňování (např. každých 5 minut), a to tak, že se současný soubor uzavře pro zápis a vytvoří se nový. Tento způsob ukládání lze považovat za jeden z nejjednodušších, nejrychlejších a také potenciálně prostorově nejúspornějších. Byť pro ukládání se jeví jako nejvhodnější, při pozdějším dohledávání jednotlivých záznamů a jiném manipulaci je kvůli absenci jakéhokoliv uspořádání nutné projít všechny záznamy od počátku souboru, což má znatelný negativní rychlostní dopad. Existuje celá řada alternativních přístupů např. použití sloupcových databází [19], které jsou při ukládání o něco komplikovanější, leč následná práce s daty při dotazování je znatelně efektivnější.

[4] *nfdump* je sada nástrojů pro sběr záznamů toků a dotazování nad nimi. Stránky projektu: github.com/phaag/nfdump

Kvůli cennému potenciálu skrytému v záznamech o tocích pouhé ukládání dnes nestačí. V záznamech je kromě běžné legitimní komunikace skryto i chování porušující bezpečnostní pravidla sítě a ohrožující bezpečnost uživatelů. Například je možné pomocí specializovaných nástrojů používajících detekční algoritmy automatizovaně identifikovat útočníka, který se snaží zmapovat zařízení a služby dostupné v síti, na kterou následně útočí. Od kolektoru se tak dá očekávat, že bude schopen předat data nástrojům, jenž takovou analýzu provozu provádějí. Předání může probíhat buď nepřímo skrze jisté formy souborů nebo databází, jak bylo zmíněno výše, nebo přímým proudovým přeposíláním. Pokud budeme uvažovat ilustrativní případ mezikroku v podobě nfdump souborů, jsou záznamy nedostupné, dokud nedojde u uzavření souboru. To v praxi znamená, že maximální prodleva mezi uložením a následnou analýzou záznamů odpovídá intervalu obměňování souborů (např. zde uvedených 5 minut). Tato dodatečná prodleva může způsobit, že útočník v době detekce již útok dávno ukončil. Navíc tento přístup dávkového zpracování souborů může zapříčinit, že záznamy nutné pro detekci sledovaných incidentů se rozprostřou mezi sousedící soubory a algoritmus je nemusí být schopen detekovat. Přímé proudové přeposílání uvedenými problémy netrpí, ale vyžaduje, aby detekční algoritmus dokázal pracovat nad průběžně přicházejícími daty např. použitím klouzavého okna. Jako zástupce nástrojů proudového zpracování lze uvést systém pro analýzu síťového provozu a detekci anomálií Nemea [22].

# Kapitola 3

# Protokol IPFIX a jeho struktura

Protokol IPFIX definuje způsob, jakým mohou být informace nejen o tocích předávány z exportérů (měřící sondy, směrovače a jiná zařízení) na kolektor. Exportér a kolektor mohou být zařízení od zcela odlišných výrobců a univerzální komunikační protokol zajišťuje, že při dodržení definovaného standardu budou vzájemně interoperabilní. Na úvod je třeba podotknout, že se nejedná o běžnou síťovou architekturu komunikace klient-server používanou na sítích, neboť veškerá komunikace je pouze jednosměrná ve směru od exportéru ke kolektoru. Protokol má binární strukturu a klade důraz na co nejvyšší prostorovou efektivitu. Zároveň vzhledem ke svému velice univerzálnímu návrhu, jak bude uvedeno dále, je možné se setkat s jeho nasazením i mimo oblasti měření toků [15].

Pro pochopení hlubších souvislostí bude v rámci této kapitoly stručně rozebrán vývoj protokolu v souvislosti s protokoly, na které historicky navazuje. Dále budou přehledově rozebrány datové struktury použité pro formátování nejen záznamů o tocích a způsob jakým jsou identifikovány vlastnosti, které byly u toků sledované a jsou přítomné v přenášených záznamech. V závěru budou popsána novější rozšíření protokolu, jenž mají nezanedbatelný vliv na přístup ke zpracování záznamů z pohledu kolektoru.

## 3.1 Historie a vývoj protokolu

Počátky snahy o měření toků sahají až do roku 1991, kdy byly poprvé sepsány obecné požadavky a cíle [11]. Kvůli tehdejšímu nezájmu ze strany výrobců síťových zařízení ovšem nedošlo k dalšímu rozvoji, byť se objevovaly další snahy o vzkříšení[1].

K prvnímu reálnému nasazení technologie sledování toků výrazně přispěla až firma Cisco tak, že vyvinula vlastní proprietární řešení označované jako NetFlow a k tomu odpovídající protokol NetFlow v5, který přenášel záznamy o tocích. V těchto ranných dobách dovoloval uvedený protokol přenášet pouze pevnou strukturu záznamu toků, což bylo z pohledu práce s daty velmi přívětivé, ale sledovatelné vlastnosti byly omezeny pouze na to, co bylo možné protokolem přenést. Pevná struktura záznamů se brzy ukázala jako značně limitující, což podnítilo vznik otevřeného protokolu NetFlow v9, kde toto omezení padlo a struktura záznamů byla navržena variabilním způsobem za pomoci šablon. S postupem času se metoda sledování toků poměrně rozšířila, vzrostl zájem mezi správci sítí a v důsledku to vedlo ostatní výrobce síťových prvků k implementaci a využití de facto stejného protokolu s případnými nekompatibilními úpravami.

[1] Pracovní skupina RTFM při IETF (viz datatracker.ietf.org/wg/rtfm)

Následná snaha vytvořit standardizovaný a univerzální protokol společný napříč všemi výrobci vyústila ve vznik IETF pracovní skupiny IPFIX (IP Flow Information Export)[2], které cílila na odstranění vzniklých nekompatibilit. Proces započal definicí obecných požadavků [14] a analýzou existujících protokolů [9]. Z analýzy vzešlo doporučení vytvořit protokol, který bude postavený na základech úspěšného NetFlow v9, avšak bude splňovat nově vytyčené požadavky. V roce 2008 světlo světa spatřil návrh standardu IPFIX pod označením RFC 5101 [2], který byl následně přetvořen do finální podoby standardu RFC 7011 [4] v roce 2013 a představuje uznávaného nástupce původního NetFlow protokolu. Jeho vývoj tím neskončil a postupně přibývala rozšíření přidávající další funkcionality. Významným důsledkem toho, že protokol IPFIX vychází z NetFlow a oba jsou současně jednosměrné protokoly je fakt, že s oběma lze pracovat velmi podobným způsobem nebo zprávy posílané pomocí NetFlow lze před interpretací na kolektoru relativně jednoduše transformovat do podoby IPFIX.

## 3.2 Informační elementy a datové typy

Záznamy přenášené v protokolu mohou mít různou strukturu v závislosti na tom, jaké vlastnosti toků exportér měří. Aby vždy bylo jasné, jaké položky se v záznamu nachází a jak se mají interpretovat, definuje standard [5] tzv. *informační elementy* (angl. *Information Elements*). Svůj informační element tak má např. zdrojová IP adresa, celkový počet paketů toku apod. Každá definice se povinně skládá z názvu, unikátního identifikačního čísla, datového typu, popisu a stavu platnosti. Volitelně navíc může být rozšířena o definici sémantiky, měrných jednotek, rozsahu platných hodnot apod. Dostatečně obecné a široce použitelné informační elementy napříč výrobci jsou spravovány ve **společném rozsahu** pod záštitou organizace IANA (Internet Assigned Numbers Authority), která zajišťuje také jejich zveřejnění. V rámci seznamu[3] lze nalézt nejen položky pokrývající vlastnosti toků extrahovatelné z různých vrstev síťového modelu, ale i položky popisující provozní parametry exportéru a jiné. Příklady vybraných elementů a přehled vybraných oblastí, které pokrývají, je uveden v tabulce 3.1. Za zmínku stojí skutečnost, že prvních 128 elementů seznamu kopíruje v zájmu zachování přenositelnosti identifikátory používané v NetFlow v9.

Jelikož přidání nových elementů do společného seznamu podléhá standardizačnímu procesu a v případě tvůrců exportérů mohou mít některé elementy experimentální charakter nebo jsou příliš specifické, mají tvůrci možnost si požádat výše jmenovanou organizaci o vlastní **privátní rozsah**. Ten je identifikován pomocí přiděleného čísla PEN (Private Enterprise Number)[4] a v rámci svého rozsahu si tvůrce informační elementy libovolně spravuje, přičemž nemá povinnost je nikde veřejně uvádět. Pro standardně definované elementy spravované organizací IANA je vyhrazen PEN s číslem 0.

Předpokládá se, že kdokoliv, kdo pracuje se záznamy toků, má představu o existenci informačních elementů a zná definici podporované podmnožiny. V rámci samotného protokolu tak standardně nedochází k přenosu definic a zúčastněným stranám musí být známé ještě před zahájením komunikace. Zatímco na straně exportéru to obvykle nevadí, jelikož mu zpravidla stačí znát jen ty elementy, které je schopen měřit a exportovat, u kolektoru je situace poněkud jiná. Kolektor by měl znát ideálně definice všechny, aby dokázal jakékoliv vlastnosti interpretovat, ale kvůli necentralizované správě privátních rozsahů to není možné.

[2] Přehled činností skupiny je dostupný na: datatracker.ietf.org/wg/ipfix

[3] Aktuálně platné informační elementy jsou dostupné na: www.iana.org/assignments/ipfix

[4] Seznam registrovaných čísel je dostupný na:
www.iana.org/assignments/enterprise-numbers/enterprise-numbers

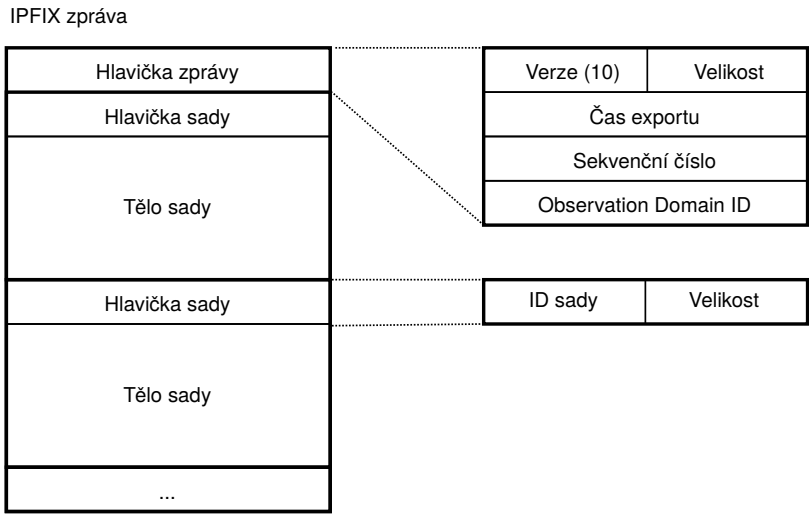
| Oblast | Název informačního elementu | Stručný popis |
|---|---|---|
| Aplikační vrstva paketů | httpContentType<br>applicationName | Typ obsahu z hlavičky HTTP protokolu<br>Název aplikace |
| Transportní vrstva paketů | sourceTransportPort<br>tcpControlBits | Zdrojový transportní port<br>TCP příznaky zaznamenaných paketů |
| Síťová vrstva paketů | sourceIPv4Address<br>ipVersion | IPv4 adresa z hlavičky paketu<br>Verze IP adresy |
| Vrstva síťového rozhraní paketů | destinationMacAddress<br>vlanId | Cílová MAC adresa<br>Identifikátor VLAN na rozhraní |
| Odvozené vlastnosti | flowStartMilliseconds<br>flowEndReason | Čas příchodu prvního paketu toku<br>Důvod ukončení toku |

Tabulka 3.1: Ukázka informačních elementů ze společného rozsahu IANA

Protože na něj mohou proudit data z několika různých exportérů od odlišných tvůrců, je vhodné, pokud alespoň podporuje přidání definic samotným uživatelemv rámci konfigurace.

Každý informační element má svůj odpovídající **datový typ** vycházející z oficiálního číselníku podporovaných typů. Mezi standardně podporované typy patří běžně očekávaná znaménková a neznaménková celá čísla, čísla s plovoucí řádovou čárkou, booleovská hodnota, časové značky a textový řetězec. Vzhledem k využití v síťovém prostředí jsou přítomny i typy pro IPv4, IPv6 a MAC adresy. Okrajovou záležitost tvoří typ pro bytové pole, který se používá zejména v situacích, kdy uvedené typy nejsou vhodné pro převod na textovou reprezentaci nebo není znám datový typ.

Při práci s datovými typy v IPFIX zprávách je nutné dbát na několik faktů. Asi nejvýraznější je zvolená datová endianita, jinak řečeno, pořadí uspořádání bajtů u políček záznamů daného datového typu. U binárních síťových protokolů – IPFIX není výjimkou – se používá tzv. big-endian často také označovaný jako network byte order, kdy na adresové místo nejnižší adresy se uloží nejvýznamnější bajt a za něj se ukládají další až po nejméně významný bajt. Avšak kolektory, které jsou spuštěny na běžných serverových procesorech, běžně používají opačný způsob kódování tzv. little-endian. Při interpretaci dat se tudíž musí dbát ohled na použitou procesorovou architekturu a data případně konvertovat. Z důvodu tlaku na co nejvyšší datovou úspornost protokolu, trochu nečekaná situace také nastává v případě celočíselných datových typu se *znaménkem* (angl. *signed*) a *bez znaménka* (angl. *unsigned*), které se vyskytují u definic informačních elementů v podobě variant zabírající 8, 16, 32 nebo 64 bitů. Neoznačují ovšem exaktně očekávanou velikost políčka záznamu, jak by se dalo předpokládat, ale maximální možnou velikost. Políčko záznamu příslušející např. datovému typu `unsigned64`, tak může být uloženo na 8, 16, 32 nebo 64 bitech, ale také na 24, 40, 48 nebo 56 bitech. Obdobná situace je také v případě čísel s plovoucí řádovou čárkou. Exportér se může, ale nemusí, rozhodnout takovýto přístup použít v případech, kdy ví, že datový typ nepoužije v plném rozsahu. Poněkud méně je též příjemná práce s typy časových značek. Existují hned 4 varianty v závislosti na přesnosti hodnoty v rozsahu od sekund pod nanosekundy. Zarážející je skutečnost, že sekundy a milisekundy vychází z formátu unixových časových značek počítající rozdíl v odpovídající přesnosti vůči 1.1.1970, zatímco mikrosekundy a nanosekundy si pro inspiraci berou formát používaný v NTP [12] s rozdílem vůči 1.1.1900.




Obrázek 3.1: Hlavička zprávy a navazující sady

## 3.3 Datové struktury použité v protokolu

Jasně definovaná struktura protokolu je nezbytně nutná pro správné předávání dat mezi exportérem a kolektorem a znalost důsledků jejího návrhu je nepostradatelná pro efektivní přístup k datům i návrh následného programového rozhraní. Při práci s datovými strukturami je stejně jako u datových typů nutné mít na paměti, že vše je kódováno ve formátu big-endian a případně provádět konverzi na požadovanou procesorovou architekturu. Na nejvyšší úrovni se binární protokol IPFIX skládá ze vždy přítomné uvozující hlavičky, na kterou navazuje řetězec sad (šablonových nebo datových) záznamů, jak je uvedeno na obrázku 3.1.

**Hlavička protokolu** na prvních 2 bajtech obsahuje zakódované číslo verze, aby šlo zřetelně odlišit IPFIX od NetFlow. Pro IPFIX v současné podobě je vyhrazena hodnota 10. Zajímavostí je, že NetFlow v5 a v9 používají hodnotu 5, respektive 9, což je i jeden z důvodů proč se IPFIX někdy nesprávně označuje jako NetFlow v10. Hlavička obsahuje také informace o celkové délce zprávy, exportním čase, sekvenčním čísle a *identifikátoru oblasti monitorování* ODID (angl. *Observation Domain ID*). Jelikož architektura měření předpokládá, že na straně exportéru může běžet více samostatných měřících procesů, které jsou schopny měřit různé body sítě (např. provoz na síťovém médiu nebo vybraných rozhraních síťového prvku) a následně se tyto záznamy předávají na odeslání společnému exportnímu procesu, používá se ODID pro odlišení oblasti monitorování v rámci daného exportéru. Je také doporučeno, aby identifikátor byl unikátní napříč všemi zařízeními, a tudíž umožnil správci sítě snadno rozeznat u dat oblast monitorování. Sekvenční číslo nevyjadřuje pořadí zprávy, jak by se na první pohled mohlo zdát, ale počet odeslaný datových záznamů od počátku komunikace. Jinak řečeno, číslo je vždy inkrementováno o počet datových záznamů přítomných v předchozí zprávě. V případě ztráty jedné nebo více zpráv při přenosu lze pomocí něj odvodit kolik takových záznamů se ztratilo. Značná komplikace nastává v okamžiku, kdy pro přenos IPFIX zpráv je použit nespolehlivý transportní protokol, který může pořadí zpráv na příjmu kolektoru zaměnit a evokovat tak falešné zdání ztracených záznamů.

Za hlavičkou se nachází sekvence sad. Každá sada je uvozena hlavičkou obsahující *identifikátor sady* (angl. *Set ID*), jenž určuje druh jejího obsahu, a její celkovou velikost v bajtech,

| Druh | ID sady | Význam |
|---|---|---|
| Template Set | 2 | Sada šablon toků |
| Options Template Set | 3 | Sada šablon nastavení a provozních parametrů exportu |
| Data set | $\geq 256$ | Sada datových záznamů |

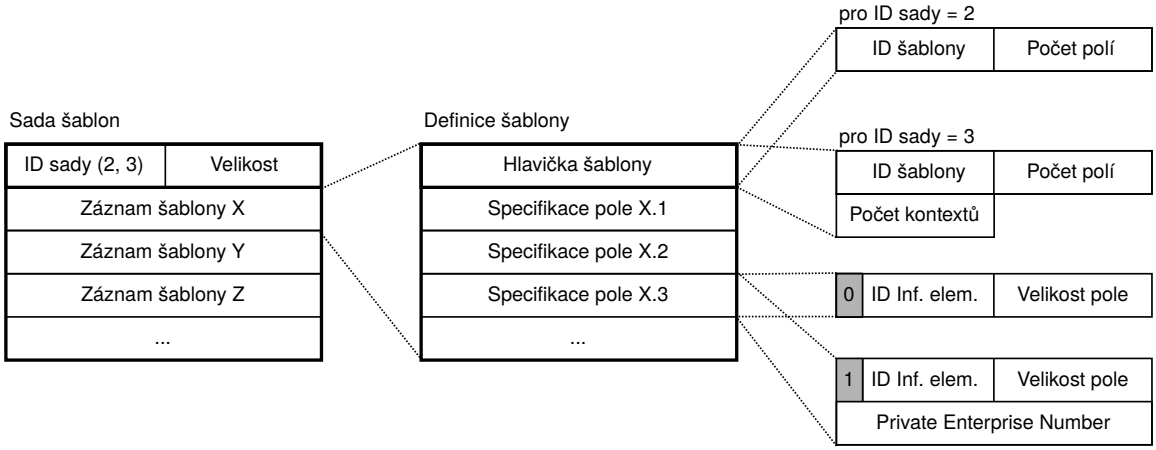
Tabulka 3.2: Sady v protokolu IPFIX

čímž umožňuje snadno určit začátek i konec sady a elegantně přeskakovat při zpracování na následující sadu. Protokol rozlišuje 3 druhy sad, jak je uvedeno v tabulce 3.2. V rámci budoucích rozšíření je možné, aby byly v případě nutnosti definovány nové.

Jedna z druhů sad, která může za hlavičkou následovat, je *sada šablon záznamů toků* (angl. *Template Set*) a *sada popisující záznamy nastavení a provozních parametrů exportéru* (angl. *Options Template Set*). Obě sady za hlavičkou společnou pro všechny sady IPFIX zprávy obsahují neprázdný seznam definic šablon odpovídajícího druhu. Rozdílem těchto dvou sad je snaha pouze odlišit šablony záznamů toků od šablon záznamů pomocných, avšak samotná struktura těchto sad je téměř totožná, liší se pouze minimálně, a proto bude o obou dále mluveno jako o sadě šablon.

Obecným **cílem šablon** je popsat z jakých jednotlivých položek se příslušející datové záznamy skládají, jejich uspořádání a velikost, kterou v záznamu zaujímají. Struktura definice šablony se skládá z hlavičky a seznamu položek, jenž obsahuje. V hlavičce se nachází identifikační číslo šablony ($\geq 256$) a informace o počtu položek seznamu. V případě šablon nastavení a provozních parametrů (Options Templates) se navíc přidává informace o tom, kolik prvních položek definuje kontext záznamu. Pořadí položek v seznamu je závazné a odpovídá pořadí odpovídajících políček v datovém záznamu tak, jak za sebou následují. Každá položka uvádí identifikační číslo odpovídajícího informačního elementu, kterému políčko záznamu odpovídá, a skutečnou délku, jenž bude políčko zabírat v záznamu. Protože se předpokládá primárně využívání společných elementů, jako výchozí rozsah identifikátorů informačních elementů se uvažuje rozsah společný (hodnota PEN 0). V případě jiných elementů z privátních rozsahů je nejvyšší bit číselného identifikátoru nastaven, což značí, že položka seznamu je prodloužena o identifikátor privátního rozsahu PEN. Příjemným důsledkem toho, že velikost políčka záznamu je uvedena přímo v šabloně a není závislá na konkrétním informačním elementu a jeho datovém typu, je, že pokud element není znám při interpretaci záznamu na kolektoru, může být políčko přeskočeno a zbytek záznamu zůstane stále čitelný. Obrázek 3.2 shrnuje uvedenou strukturu sady i šablon. Co se ještě týče hodnoty velikosti, v případech textových řetězců i některých jiných datových typu často není předem známo, kolik místa bude políčko vyžadovat. Pokud by byly povoleny položky pouze fixní délky, často by to pro řetězce znamenalo nutnost zkrácení nebo plýtvání vyhrazeným místem. Jako velikost je z toho důvodu možné uvést rezervovanou hodnotu 65535 (tj. $2^{16} - 1$), která značí, že se jedná o tzv. položku dynamické délky a skutečná délka je uvedena až jako prefix položky v datovém záznamu.

Šablony popisují buď striktně **statický záznam** neobsahující žádné položky dynamické délky, nebo **dynamický záznam** s alespoň jednou dynamickou položkou. Z toho plynou poměrně významné důsledky pro práci s daty, neboť u šablony statického záznamu je možné si předem vypočítat, jak je celkový záznam dlouhý a na jaké relativní pozici vůči počátku záznamu se které políčko nachází. Z pohledu práce se záznamem to dokáže



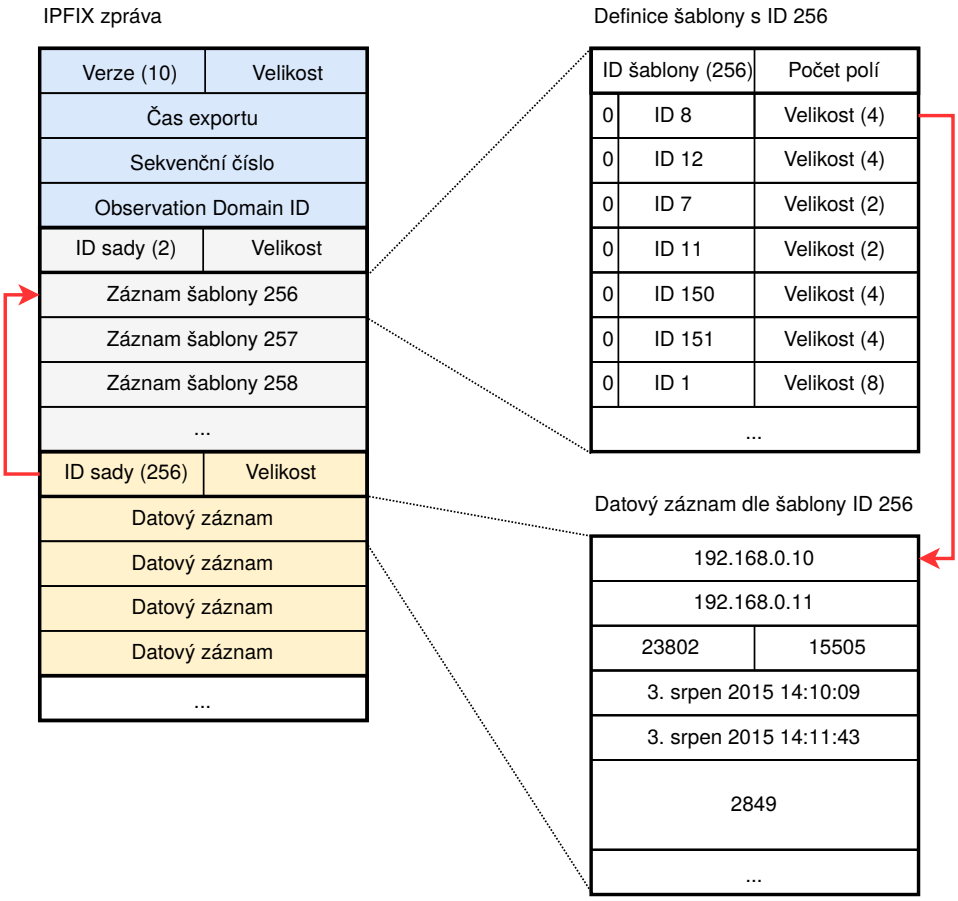

Obrázek 3.2: Struktura sady šablon

velmi usnadnit a zrychlit práci, jelikož při požadavku na čtení konkrétní položky stačí znát začátek záznamu, relativní pozici položky a její velikost. V okamžiku, kdy alespoň jedna položka záznamu je dynamické délky, už dochází ke komplikacím. Nelze předem určit velikost odpovídajícího záznamu ani relativní pozici všech položek vzhledem k tomu, že statické a dynamické položky se mohou libovolně míchat. Relativní pozice je tak známá pouze u položek nacházejících se na začátku až po výskyt první dynamické včetně. To značně komplikuje přímý přístup k ostatním položkám.

**Zpracování datových sad** na rozdíl od šablonových je značně komplikovanější už jen tím, že struktura není pevná a vyžaduje znalost odpovídající šablony, která sadu popisuje. Zatímco sady šablon mají v hlavičce pevné ID určující, o jakou sadu se jedná (ID 2 nebo ID 3), v případě datových sad tomu tak není, neboť pro ně jsou vyhrazena ID $\geq 256$. Podle příslušné hodnoty ID u konkrétní datové sady se při interpretaci dohledá dříve definovaná šablona se shodným identifikačním číslem a podle této šablony jsou zakódovány všechny záznamy v sadě. Záznamy jsou uloženy v řadě za sebou s tím, že neobsahují žádnou oddělující hlavičku. V případě, že záznam je statický, lze snadno ze znalosti šablony vypočítat pevnou délku jednotlivých záznamu, určit kolik záznamů se v sadě nachází a také jejich počáteční pozice. Neexistence hlaviček ovšem přináší komplikaci v případě záznamů dynamické délky. Aby se při zpracování sady těchto záznamů určila skutečná délka každého záznamu v sadě, je nezbytné projít skrze všechna jeho dynamická políčka, což je také jediný způsob, jak zjistit, kde začíná navazující záznam. Vedlejším efektem neexistujících hlaviček záznamů je i fakt, že pokud není známá šablona pro interpretaci datové sady, nelze ani zjistit, kolik záznamů se v sadě nachází a bylo ignorováno.

## 3.4 Správa šablon a vliv transportních protokolů

Obrázek 3.3 názorně naznačuje, jak se šablony používají pro interpretaci datových záznamů. Posílat šablony pro interpretaci dat v každé zprávě by bylo poměrně nelogické ať už z hlediska nároků na množství zabraného místa nebo výpočetních prostředků vynaložených na jejich opakované zpracování. Jednoduše stačí, aby šablony byly definovány pouze jednou a všechny navazující zprávy už mohou obsahovat pouze datové sady odkazující se na dříve popsané šablony z předešlých zpráv. Obvykle se všechny definice šablon hromadně pošlou na začátku ihned po navázání komunikace s kolektorem. Meze se přirozeně nekladou, aby




Obrázek 3.3: Příklad IPFIX zprávy s návazností na šablony

se později v okamžiku nutnosti definovaly šablony další. Z toho vyplývá, že ten, kdo bude proud IPFIX zpráv chtít interpretovat, si musí průběžně udržovat seznam šablon, které byly definovány, a musí být schopen v nich vyhledávat. Co lze definovat, jde i zrušit. Když exportér ví, že šablonu nepotřebuje, může ji odebrat a to tak, že pošle její novou definici obsahující nulové množství položek záznamu. Od té chvíli se už žádné navazující datové sady na šablonu nesmí odkazovat. Co bylo zrušeno, je přirozeně možné znovu použít. Identifikační číslo šablony může exportér znovu upotřebit pro šablonu novou a její definice může být zcela odlišná. Z toho plyne – a to je velmi důležité – že v průběhu času se pod stejným ID šablony mohou nacházet jiné definice a rozsah období platnosti šablony je dán časem jejího prvního pozorování a dobou, kdy přestala platit nebo byla nahrazena. Správu šablon není radno podceňovat.

Zde je nutné opětovně upozornit, že komunikace pomocí protokolu IPFIX je pouze jednosměrná, tj. od exportéru na kolektor, který skrze protokol IPFIX tedy nemá možnost s exportérem jakkoliv přímo komunikovat a ani mu sdělit, že šablony přijal nebo že mu chybí. Výše uvedený postup s definováním, rušením a dohledáváním šablon pro interpretaci datových sad funguje dle očekávání pouze za podmínek, kdy je garantováno, že všechny odeslané zprávy příjemce obdrží a jsou přijaty ve stejném pořadí, jak byly odeslány. Například při ztrátě zprávy obsahující definice šablony by hrozilo, že nebudou datové sady interpretovány nebo že následně přijde požadavek na zrušení neznámé šablony.

Za samotný přenos zpráv od exportéru ke kolektoru přebírá odpovědnost **transportní protokol** TCP, UDP nebo SCTP (viz podkapitola 2.4). Avšak garanci spolehlivosti a pořadí doručení je schopen vždy zaručit pouze transportní protokol TCP. U ostatních dvou může docházet ke ztrátám i změně pořadí a z těchto důvodů jsou v případě jejich použití standardem [4] upravena pravidla chování a tvorby IPFIX zpráv. Pro SCTP přenos stačí pouze zajistit, aby zprávy obsahující definice a rušení šablon chodily spolehlivě a s garantovaným pořadím. Zprávy obsahující pouze datové sady už mohou, ale nemusí, chodit nespolehlivě i mimo pořadí. Tohle jsou ovšem drobnosti ve srovnání se změnami platnými pro UDP, které není schopné v podstatě nic garantovat. Aby byla zajištěna alespoň jistá míra jistoty, že kolektor bude mít definice šablon i v případě ztráty některých zpráv protokolem UDP, je doporučené, aby je exportér při zahájení komunikace poslal vícekrát v samostatných zprávách. Navíc v pravidelných časových intervalech (nebo i každých N paketů) musí všechny šablony opětovně definovat, a to z důvodu zakázání mechanismu rušení šablon a jeho nahrazení jejich omezenou časovou platností. Každá šablona, která přijde prostřednictvím UDP na kolektor, a nebyla po stanovenou konfigurovatelnou dobu přijata, může být považována za již neplatnou a kolektor s ní již nemusí počítat. Poněvadž kvůli zákazu přímého rušení šablon exportér nemůže šablonu přímo zrušit, měl by počkat alespoň trojnásobek intervalu přeposílání, než stejné ID šablony znovu použije pro jinou definici. Paradoxně dle standardu ale kolektor musí každou novou definici ihned akceptovat a nahradit tu původní, i když ta z jeho pohledu může stále platit.

Pozornému oku neunikne, že i přes uvedená opatření u SCTP a UDP stále **hrozí přeskládání** pořadí datových zpráv. Pokud hypoteticky dojde ke změně definice šablony a zároveň následně přijde opožděná zpráva s datovými záznamy, které takovou šablonu používaly, hrozí, že kolektor by se mohl pokusit použít novější šablonu na dekódování záznamů tvořených podle šablony starší. Buď by kolektor tuto příhodu mohl rozeznat, kvůli nesedící délce datového bloku a neočekávanému konci jednoho ze záznamů, nebo by data bez povšimnutí zcela špatně interpretoval. Aby se tomu předcházelo, exportér by měl použít ochranné časové limity, než znovu šablonu užije. Kolektor má být v této oblasti také proaktivní a dle standardu může implementovat vyrovnávací paměť příchozích zpráv a použít informace o exportní času z jejich hlaviček k určení uspořádání akcí při správě šablon.

Za pozornost také stojí způsob **vypořádání se s přerušením spojení** nebo situací, kdy kolektor zahájí svou činnost až po exportéru, kterému se tudíž na počátku nemohlo podařit navázat spojení. Transportní protokoly TCP a SCTP umožňují zjišťovat, zda druhá strana komunikace existuje a data přijímá. Pokud tomu tak není, jsou schopny na tuto událost upozornit. Exportér se tak může pokoušet navázat spojení opakovaně a v případě úspěchu postupovat jako při novém spojení, a tedy poslat prve všechny šablony a pak už jen datové sady. Funguje to i opačně, kdy se kolektor dozví, že exportér již spojení ukončil, čímž může všechny prostředky pro něj vyhrazené uvolnit a zapomenout všechny jeho šablony. Do hry opětovně vstupuje UDP, které tohle nepodporuje. Exportér nemůže s jistotou vědět, zda jej vůbec nějaký kolektor poslouchá, a to je i důvod, proč musí neustále v pravidelných intervalech posílat definice stále platných šablon. Kolektor, který totiž zahájí svou činnost a příjem až po exportéru, se dostane do situace, kdy zmeškal úvodní definice šablon a přicházejí mu už jen zprávy s datovými sadami, jenž přirozeně není schopen bez znalosti šablon interpretovat. Nijak nemůže na tuto skutečnost exportér upozornit a může jedině čekat, než exportér znovu pošle šablony v rámci pravidelných intervalů. Do té doby může všechny zprávy zahazovat. Mohl by je sice dočasně uchovávat a pak je podle později přijatých šablon interpretovat, ale vystavuje se riziku, že odpovídající šablony mohly být v mezidobí změněny. Navíc kolektor také u UDP není schopen detekovat, že exportér svou

činnost ukončil. Existenci spojení s exportérem by tak měl svázat s neaktivním časovým limitem, po jehož uplynutí jsou příslušející vyhrazené zdroje uvolněny.

Exportér může provádět měření na různých oblastech sítě a k určení původu se v hlavičce každé IPFIX zprávy uvádí identifikační číslo ODID. V rámci různých oblastí může hypoteticky probíhat měření mírně odlišných vlastností toků, a i z tohoto důvodu je platnost šablon vázaná na konkrétní ODID. Jinak řečeno, šablona z jednoho ODID může být použita pouze pro dekódování datových sad ze stejného ODID a v různých ODID se vyskytují na sobě zcela nezávislé šablony i se stejným identifikačním číslem. Zároveň kolektor často přijímá data od více samostatných exportérů, mezi nimiž není žádné spojení a které si definice šablon řídí zcela nezávisle na ostatních. Kolektor proto musí odlišovat také šablony a data z různých transportních spojení s exportéry. Když se to shrne, **kontext platnosti šablon** je vázán na kombinaci transportního sezení a ODID, ve kterém byly definovány. Interpretace libovolné datové sady tak začíná v určení kontextu a následném dohledání šablony v jeho rámci. Předešlé odstavce této podkapitoly charakterizují činnosti nad šablonami v rámci jednotlivých kontextů.

Z výše uvedeného lze vyextrahovat několik zásadních sdělení. Rozsah platnosti každé šablony je unikátní v rámci kombinace ODID a transportního spojení. Vyskytují se zpravidla v prvních zprávách komunikace nebo u UDP také během pravidelných obnov. V průběhu času mohou být definovány, rušeny a předefinovány. Protokol IPFIX je ovlivněn zvoleným transportním protokolem a v závislosti na protokolu hrozí, že část zpráv bude ztracena nebo dojde k přeskládání pořadí. Ztrátě zabránit přímo nelze, ale u přeskládání je třeba aplikovat mechanismy bránící použití špatných šablon.

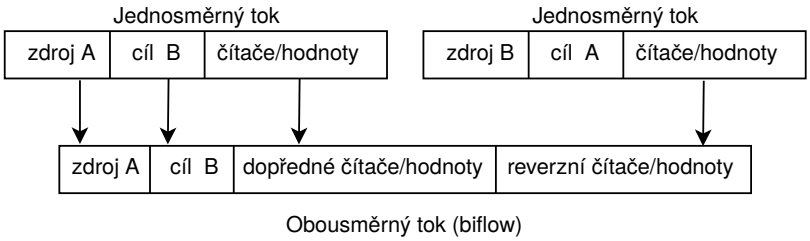
## 3.5 Vybraná rozšíření protokolu

S nasazením a rozšiřováním protokolu IPFIX se ukázalo, že pokročilejší požadavky nejsou základním standardem pokryty. Vznikla celá řada další návrhů standardů, které původní protokol rozšiřují o nové schopnosti. Významné je, že všechna rozšíření mají jedno společné. Pokud je kolektor přímo nepodporuje, nebrání mu to v práci s daty. Čemu nerozumí, to může ignorovat bez fatálních dopadů. Rozšíření existuje celá řada, ale zde budou uvedena pouze ta nejvýznamnější, která mají značný dopad na následný návrh kolektoru a mají do budoucna významný potenciál.

### Obousměrné toky

Již při popisu samotné definice toku (podkapitola 2.2) bylo zmíněno, že existuje i záznam obousměrného toku tzv. biflow, který v sobě skloubí oba směry komunikace a do analýzy přináší mnohem bohatší informační hodnotu než dva samotné a těžko spárovatelné jednosměrné záznamy. Rozšíření je specifikováno v rámci návrhu standardu RFC 5103 [17] a je poměrně jednoduché. Datový biflow záznam je z běžného pohledu k nerozeznání od běžného záznamu v protokolu IPFIX. Rozdíl tkví v asociovaných informačních elementech. Dosud uváděné elementy popisující položky záznamu lze charakterizovat jako dopředné (jdoucí od zdroje k cíli) a nově k nim přibyly elementy reverzní (směřující od cíle ke zdroji). Reverzní informační elementy jsou de facto kopií běžných elementů a vyjadřují navíc pouze opačný směr. Pokud například IPFIX element vyjadřující počet paketů zahrnutých v toku od zdroje k cíli se označuje jako *packetDeltaCount*, reverzní element může být pojmenován jako *packetDeltaCountReverse*[5] a vyjadřuje úplně stejnou vlastnost jen v opačném směru.

[5] Názvosloví není standardem striktně definováno.




Obrázek 3.4: Konceptuální diagram složení biflow záznamu [17]

Struktura složení záznamu je poměrně jednoduchá. Pro oba směry platí, že klíčové vlastnosti toku (např. typická 5tice) jsou v podstatě stejné a pouze směrové položky jsou z pohledu reverzního směru prohozeny (např. zdrojová IP adresa je cílová a naopak). Aby tak nedocházelo ke zbytečné duplikaci dat, biflow záznam obsahuje klíčové hodnoty toku pouze pro dopředný směr. Součástí záznamu jsou následně neklíčové hodnoty pro dopředný směr popsané také pomocí běžných informačních elementů a neklíčové hodnoty pro reverzní směr popsané ovšem pomocí reverzních informačních elementů. To je z pohledu rozšíření v podstatě vše.

### Seznamy a strukturované datové typy

Každý datový záznam přenášený skrze protokol IPFIX ve standardní verzi je ve své podstatě jednoduchá struktura, která nemůže obsahovat jiné strukturované typy jako pole nebo seznamy, což může být za jistých okolností na škodu. Oblastí potenciálního využití může být např. snaha o zachycení seznamu VLAN itentifikátorů nebo strukturované odlišení aplikačních rozšíření od základních vlastností záznamu toku.

Strukturované typy mají význam, a proto vzniklo rozšíření v podobě návrhu standardu RFC 6313 [3], který přidává podporu pro nové datové typy, jenž je nyní možné přiřadit informačním elementům:

- **basicList** Seznam položek jednoho informačního elementu
- **subTemplateList** Seznam struktur popsaných dle společné šablony
- **subTemplateMultiList** Seznam struktur popsaných dle odlišných šablon

V rámci zachování kompatibility se struktury šablon nijak nemění a veškeré změny se uskutečnily až na úrovni datových záznamů. Kolektor, který strukturovanou položku záznamu interpretuje, nejprve dle definice informačního elementu zjistí, o který nový datový typ se jedná a podle toho bude dále postupovat. Pokud daný typ nezná nebo nezná ani odpovídající informační element, může přeskočit strukturovanou položku dle znalosti její délky z šablony. Všechny nové strukturované typy jsou zakódovány tak, že obsahují speciální hlavičku. V případě typu `basicList` obsahuje informaci, jaký informační element seznam obsahuje. U ostatních dvou typů to je identifikační číslo, resp. identifikační čísla šablon, které se na dekódování mají použít. Za povšimnutí stojí, že při interpretaci jedné datové sady záznamů strukturovaných dle stejné šablony, je možné, aby každý záznam obsahoval v rámci stejné položky typu `basicList` pokaždé jiný informační element nebo u `subTemplateList` pokaždé odkaz na jinou šablonu. Tato skutečnost má dalekosáhlé následky, neboť pouze na základě hlavní šablony záznamu toku už není možné předem úplně určit, co ve skutečnosti obsahuje a je nutné se podívat vždy do zpracovávaných příchozích dat.

# Kapitola 4

# Kolektor současné generace a jeho analýza

Jedním ze zástupců IPFIX kolektorů je nástroj s příznačným pojmenováním IPFIXcol[1]. Důvodem výběru tohoto nástroje jako podkladu pro téma práce a budoucí generaci kolektoru byla jeho dlouhodobá znalost autorem práce, který je zároveň i přispěvatelem do tohoto projektu. Mezi přednosti kolektoru se řadí jeho dostupnost široké veřejnosti v podobě otevřeného zdrojového kódu, důraz na výkonost řešenou i prostřednictvím paralelizace a neméně důležitá rozšiřitelnost pomocí zásuvných modulů. Kolektor prošel od svého vzniku již značným vývojem, který se v některých ohledech neblaze podepsal na jeho stavu. Část klíčových komponent nebyla řádně navržena nebo v době jejich vzniku se nepředpokládalo budoucí rozšíření o další schopnosti. Také byly objeveny odchylky od korektního chování popsaného ve standardu protokolu IPFIX [4]. Všechny tyto příčiny v důsledku vedou ke vzniku nových chyb a implementace podpory nových rozšíření představených v rámci protokolu IPFIX se stala již stěží realizovatelnou.

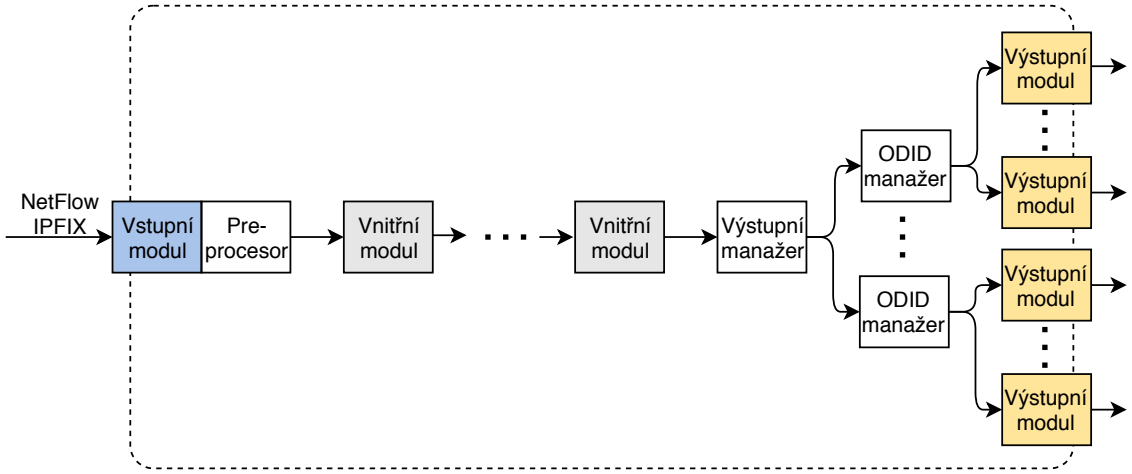
Cílem této kapitoly je představit podobu řešení současné generace kolektoru, charakterizovat činnosti jednotlivých komponent, zprostředkovat přibližný pohled na postup zpracování dat a více přiblížit úskalí současného návrhu. Kolektor a jeho komponenty budou posuzovány nejen z pohledu konceptuálního, ale i z hlediska programové stránky a její kvality.

## 4.1 Architektura kolektoru a proud dat

Základním předpokladem všech kolektorů je schopnost data přijímat z různých zdrojů skrze vybrané transportní protokoly, záznamy toků případně modifikovat a uchovávat je pro dlouhodobé uskladnění nebo poskytovat dalším nástrojům pro analýzu. Konceptuální architektura a identifikace komponent je klíčová pro alespoň povrchové pochopení postupů práce a činností, které v kolektoru probíhají při práci s daty toků.

IPFIXcol pro zpracování IPFIX zpráv a záznamů zprostředkovává rozhraní a funkce, které poskytuje třem druhům zásuvných modulů: vstupním, vnitřním a výstupním. Kromě samotného rozhraní má na starosti tvorbu zřetězeného propojení, správu a konfiguraci modulů, které mohou být dokonce za běhu kolektoru zaváděny nebo vypínány bez přerušení příjmu dat. Ve své podstatě může být na kolektor nahlíženo jako na manažera, který propojuje a řídí nezávislé moduly s cílem zpracovat záznamy toků. Obrázek 4.1 znázorňuje jakým

[1] http://github.com/CESNET/ipfixcol/




Obrázek 4.1: Architektura stávajícího kolektoru

způsobem jsou v rámci kolektoru propojeny a jak probíhá zřetězené zpracování dat. Je záhodno podotknout, že instance jednotlivých modulů, jak jsou vyobrazeny, běží zcela paralelně ve vlastních vláknech z důvodu vyšší výkonnosti. Přímo se také nabízí srovnání s obrázkem 2.4, jenž vyjadřoval obecný návrh kolektoru. Činnost jednotlivých komponent bude ilustrována na příkladu postupného zpracování IPFIX zpráv.

Jak bylo v předchozí kapitole zmíněno, data na kolektor přichází od exportéru skrze podporovaný transportní protokol. Příjem a zpracování má na starost jeden vybraný **vstupní modul**, jehož náplní je surová data se záznamy toků od exportéru přijmout a poskytnou dalším částem kolektoru. K dispozici jsou vstupní moduly pro UDP, TCP, SCTP i surové IPFIX zprávy uložené v souboru. V případě, že se jedná o zprávu formátovanou dle protokolu NetFlow, dochází k její konverzi na IPFIX zprávu, což zahrnuje mírnou úpravu hlavičky a samotných šablonových a datových záznamů kvůli mírně odlišnému formátování. Výstupem každého vstupního modulu je tak vždy IPFIXová zpráva, která postupuje do preprocesoru zpráv. Náplní preprocesoru je zkontrolovat korektnost zprávy, vyextrahovat šablony a označit, kde se záznamy ve zprávě nachází a podle jakých šablon mají být zpracovány. Nad surovou zprávou vytvoří obálku s rozhraním, se kterým mohou navazující moduly a funkce kolektoru pracovat.

Předzpracovaná zpráva v zaobalené podobě přichází na volitelný řetězec **vnitřních modulů**, jejichž náplň zahrnuje úpravu, odebírání nebo obohacování záznamů toků o nové položky. Je možné například filtrovat toky dle jejich vlastností, provádět anonymizaci IP adres toků nebo obohacovat záznamy o geolokační informace. Vybrané moduly mohou také provádět obohacení záznamů jejich označkováním, což jsou dočasné vlastnosti existující jen během zpracování, a navazující moduly mohou tyto značky používat při svém zpracování. Např. vnitřní profilovací modul provede zařazení toků do skupin dle uživatelem definovaných pravidel a výstupní modul, který tyto informace umí zpracovat, se následně těmito skupinami při ukládání řídí. Výběr nasazených modulů je pouze na uživateli kolektoru, který svým výběrem ovlivňuje, co bude s daty provedeno, a v jaké posloupnosti moduly uspořádá. V řadě případů nasazení nejsou moduly žádoucí, a proto může být posloupnost zcela prázdná.

Zprávy dále putují na výstupní manažer a ten je distribuuje **výstupním modulům**, které mohou být řazeny do separátních skupin podle toho, zda zpracovávají všechny zprávy přicházející na kolektor nebo pouze z vybrané oblasti monitorování dle identifikátoru ODID.

Moduly provádí uchování záznamů do souborů pro dlouhodobé skladování nebo je dále distribuují. Jednou z možností je ukládání do stále populárního formátu nástroje *nfdump*, který vychází ještě z původního NetFlow a dovoluje pracovat pouze se základní podmnožinu vlastností toků. Data lze konvertovat i do textové podoby formátu JSON, který je možné ukládat jak na disk, tak i distribuovat dalším aplikacím. Činnost výstupních modulů tak typicky zahrnuje transformaci, a navíc někdy způsobuje i nezvratné ořezání nepodporovaných vlastností toků.

## 4.2 Preprocesor IPFIX zpráv a správa šablon

Asi nejvíce klíčovou komponentu představuje preprocesor zpráv. Je v podstatě nepřímou součástí vstupního modulu a bezprostředně od něj přijímá zprávy, se kterými pracuje. Nad každou zprávou vytvoří **zaobalující obálku**, jenž obsahuje mimo jiné informace, který exportér je zdrojem dat, z jakého ODID data pochází apod. Jednotlivé sady zpráv preprocesor prochází od začátku do konce. Nejprve zpracuje šablony, které uchová ve správci šablon. Následně projde skrze všechny datové sady a pro každý jejich záznam vytvoří trojici (začátek záznam, jeho délka a odkaz na šablonu), kterou udržuje jako součást obálky. Tohle řešení má tu zásadní výhodu, že kdokoliv, kdo bude pracovat se zaobalenou zprávou, už si nemusí pamatovat a dohledávat šablony a už vůbec nemusí celou zprávu znovu procházet ať už obsahuje datové záznamy statické, nebo dynamické velikosti. Velice to ostatním modulům urychlí a zjednoduší přístup k jednotlivým záznamům.

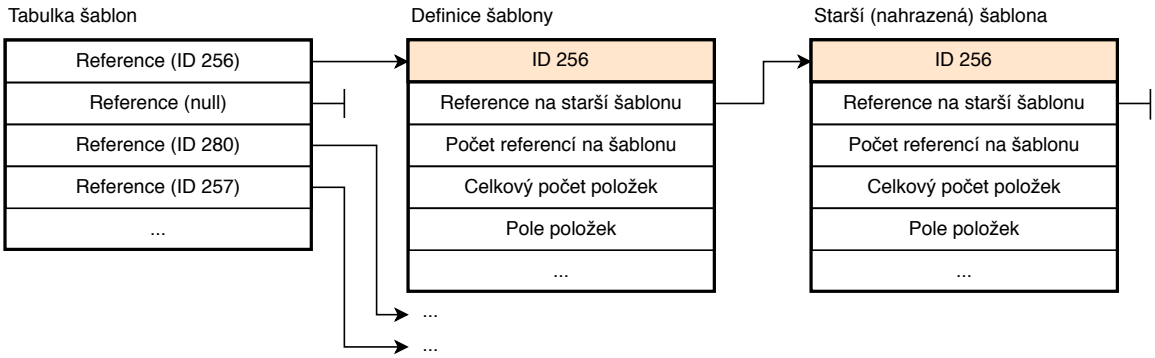
**Správa šablon** je vedlejší a neméně důležitou součástí preprocesoru. Šablony jsou zde uchovány a rušeny dle požadavků protokolu a při interpretaci datových sad se zde dohledávají. Zpracovaná šablona zde mimo jiné obsahuje její samotné ID, identifikátor druhu šablony a surový seznamu položek s identifikátory informačních elementů obsažených v příslušném datovém záznamu. Právě na tyto šablony je odkazováno z uvedené trojice. Zaobalené IPFIX zprávy, které z preprocesoru putují do dalších modulů, tak stále ukazují na šablony v preprocesoru. Pokud tedy dojde ke zrušení platnosti šablony nebo k jejímu nahrazení, není možné, aby byla okamžitě z paměti uvolněna, neboť je velká pravděpodobnost, že ji stále ještě nějaká instance dále ve zřetězené lince používá. Z těchto důvodu každá šablona obsahuje čítač referencí, který je inkrementován při vytvoření trojice v preprocesoru a dekrementován při zrušení po zpracování všemi výstupními moduly.

Nyní je vhodné upozornit na některé **chyby a nedostatky** návrhu a implementace. Použití čítačů referencí není vzhledem paralelnímu zpracování úplně šťastné. Inkrementaci a dekrementaci provádí zcela jiné části kolektoru obsluhované jinými vlákny, a navíc současně se stejnou šablonou pracují jiná vlákna vnitřních a výstupních modulů. Kvůli nevhodně zvolenému uspořádání struktury tak navíc dochází k falešnému sdílení[2]. Chyby se nevyhnuly ani samotné struktuře zaobalení zpráv. Ještě před implementací seznamu trojic existovala pole ukazatelů na jednotlivé šablonové a datové záznamy zprávy (bez návaznosti na šablony pro zpracování). Z historických důvodu byla zachována, avšak jejich velikost je značně naddimenzována. Pro každý datový záznam a oba druhy šablon zvlášť rezervují fixních 1024 ukazatelů. Pokud uvážíme 64bitovou procesorovou architekturu jen tato pole zaberou 24kB (3 x 1024 x 8B) na každou zaobalenou IPFIX zprávu. To by se na první pohled mohlo zdát jako nepodstatné, ale v kolektoru současně mohou koexistovat i tisíce těchto zpráv. O plýtvání také svědčí to, že většina obvyklých IPFIX zpráv neobsahuje více než několik

[2]Falešné sdílení je situace, kdy jeden procesor zapisuje a druhý čte z téhož bloku. Každý sice přistupuje na jinou adresu, ale vzájemně si invalidují cache pamět.

desítek záznamů toků. V několika oblastech je též chybný postup preprocesoru. Tím, že prve zpracuje všechny šablonové sady a pak teprve datové, porušuje správný postup, kdy se má postupovat postupně po jednotlivých sadách od hlavičky dále. Při nevhodném uspořádání sad hrozí vybrání špatné šablony pro interpretaci datových záznamů. Pokud také dojde k zaslání poškozené zprávy, standard praví, že má dojít k rozvázání spojení (TCP nebo SCTP) s exportérem, což současná realizace nedovoluje.

Ani správce šablon se nevyvaroval odchýlení od pravidel stanovených standardem. Za nejvýznamnější prohřešek lze považovat celkové **ignorování exportního času** zpráv. Pokud by např. došlo ke změně definice šablony a následně by přišly opožděné záznamy formátované dle starší verze šablony, nemá preprocesor možnost si starší šablonu od správce vyžádat a použije jen tu novější. Zároveň správcem není implementována podpora pro rušení všech šablon daného typu tzv. All (Options) Templates Withdrawal. Samotný způsob uchování šablon také není zrovna efektivní. Pro rozsah platnosti šablon daný kombinací transportního sezení a ODID si uchovává tabulku s referencemi na šablony, jak je uvedno na obrázku 4.2. Nejenže odkazy jsou v tabulce neuspořádané, existují v ní i prázdná místa. Při vyhledávání šablony tak nelze použít například binární vyhledávání a celá tabulka se musí vždy procházet od začátku, dokud nedojde k nalezení požadované šablony nebo jejího konce. Pokud by byla šablona navíc změněna a na její předchůdkyni se stejným ID by stále existovaly reference, jsou šablony uchovány v lineárním seznamu.




Obrázek 4.2: Správce šablon

Pro práci se strukturovanými datovými typy by se teoreticky dal použít globální správce šablon, do kterého mají i samotné moduly přístup a umožňuje jim případně dohledávat definice šablon. Nepoužitelným ho ale činní jeho globálnost, kdy na vstupu preprocesor může nahrazovat definice šablon novými, zatímco na výstupu jiný modul pracuje s daty formátovanými stále podle starých šablon. Práce se špatnou šablonou způsobená souběhem vláken by tak mohla mít ve svém důsledku až fatální následek v podobě havárie kolektoru.

## 4.3 Rozhraní zásuvných modulů a jejich zřetězení

Zásuvný modul není z pohledu kolektoru nic víc než dynamická knihovna načítaná až za jeho běhu. **Rozhraní modulu**, které kolektor používá, se obvykle skládá ze tří funkcí. Vždy je přítomna inicializační funkce volaná pro tvorbu instance modulu, který si přichystá svoje vnitřní struktury pro následující zpracování dat. Opačný účel plní finalizační funkce. Třetí funkce se významově mírně liší pro různé druhy modulů. Pro vstupní moduly znamená nařízení pro získání dat, u vnitřních pro zpracování a případnou modifikaci a u výstupních

modulů pro uložení nebo distribuci. Moduly nejenže funkcionalitu nabízí, používají zároveň rozhraní a funkce pro zpracování dat poskytované kolektorem, což i kolektor činní z jejich pohledu knihovnou. Rozhraní kolektoru i modulů je napsáno v programovacím jazyce C, avšak oba mohou být vnitřně implementovány jakkoliv jinak.

Paralelismus zpracování je v rámci kolektoru realizován prostřednictvím **zřetězené linky**. Pro každou instanci modulu je vytvořeno samostatné vlákno, které kromě řízení komunikace se sousedními moduly, volá i funkce rozhraní modulu. Pro přenos zabalených IPFIX zpráv od vstupu až na výstup jsou mezi jednotlivými instancemi vloženy fronty realizované pomocí kruhových bufferů. Aby se co možná nejvíce omezilo kopírování velký struktur, přes frontu se posílají pouze ukazatele na obálku zprávy. Po zaplnění fronty dochází k blokování činnosti zapisujícího vlákna. Pokud například nějaký výstupní modul nebude stíhat zpracovat záznamy, může se v důsledku stát, že se blokování zpropaguje až na vstup, a protože se na vstupu zablokuje vlákno řešící odebírání dat hrozí, že bude docházek ke ztrátě zpráv.

To, že se skrze frontu přenáší celá zaobalená IPFIX zpráva místo jednotlivých záznamů, má řadu výhod. Zejména se nemusí přenášet tolik dat, záznamy jsou lépe dostupné i z pohledu paměťové lokality a také se omezuje množství paměťových alokací. V okamžiku, kdy se záznamy toků obohacují o nové položky nebo se z nich naopak něco ubírá, nastává poměrně nešťastná a komplikovaná situace, protože se celý surový paket, odkazy z obálky i ukazatele na šablony musí upravit. Tohle ovšem nastává pouze ve výjimečných případech, a proto je i přenášení celých zpráv stále vhodnější. Poměrně neblahá je ovšem skutečnost, že skrze zřetězenou linku chodí pouze jediná struktura, a to ta zaobalující zprávu. Kromě samotné IPFIX zprávy, kterou zaobaluje, musí být schopna i signalizovat, že se exportér připojil nebo odpojil, neboť některé moduly si udržují pomocné struktury pro jednotlivé zdroje. Nikde ale není řádně zdokumentováno, kdy a jak jsou políčka struktury vyplněná, což přináší nejistotu při zpracování dat.

## 4.4 Způsob práce se záznamy

Zásuvné moduly až na pár výjimek obvykle spoléhají na rozhraní kolektoru a pracují s obálkou zprávy. Už bylo zmíněno, že pomocí seznamu trojic, které jsou součástí obálky, je možno k záznamům poměrně snadno přistupovat. Má-li modul zájem o konkrétní položku ze záznamu, může využít funkci, která za pomocí šablony najde výskyt položky požadovaného informačního elementu. Avšak funkce není schopna upozornit na situaci, kdy ten stejný element je přítomen v záznamu vícekrát, a vždy vrátí pouze ukazatel na případný první výskyt.

Výstupní moduly mnohdy transformují záznam toku do podoby jiného formátu. Při tom iterují přes všechny položky záznamu a snaží se je převést. Až poněkud zarážející je fakt, že neexistuje žádný oficiální **iterátor**, který by tohle zajišťoval. Kód, který byl použit v jednom z modulů byl bez značného uvážení často překopírován, což odporuje správným principům a zvyšuje riziko zanesení nechtěných chyb. Manuální iteraci komplikují zejména položky dynamické délky, u kterých je třeba dbát na velikost uloženou v samotném datovém záznamu místo v šabloně.

Další nepříjemnost je, že **struktura šablon** spravovaných správcem šablon je značně strohá. Obsahuje v podstatě pouze ty stejné informace jako definice šablony v protokolu IPFIX, tj. identifikační čísla informačních elementů a jejich velikost v datovém záznamu. Byť by se to na první pohled mohlo zdát dostačující, není tomu tak. U jednotlivých položek chybí provázání s definicí příslušného elementu. Jinými slovy, není přímo znám ani název, ani da-

tový typ a ani jiné parametry. Například takový výstupní modul konvertující záznamy toků do formátu JSON se musí s každou položkou záznamu dotazovat databáze informačních elementů, jaký element se pod daným ID skrývá. Dohledávání přirozeně nějaký čas stojí a ubírá drahocený výkon.

Jiná úskalí skýtají **datové typy a jejich kódování**. Neškodí připomenout, že položky záznamu jsou kódovány dle datové endianity big-endian, což na většině běžných počítačových procesorových architektur vyžaduje konverzi. To by nemusel být zas takový problém díky standardním konverzním funkcím, pokud by navíc u číselných hodnot, které jsou mimochodem nejběžnější, nedovoloval standard IPFIX navíc provádět kódování s redukovanou velikostí. Například maximálně 8 bajtová číselná hodnota může být ve skutečnosti uložena nejen na standardních 8, 4, 2 nebo 1 bajtu, ale i na neobvyklých 3, 5, 6 a 7 bajtech. Kolektor žádné speciální konverze pro tyto případy nenabízí a snad žádný modul si s redukovaným kódováním neporadí. Už ani nepřekvapí, že strukturované typy nejsou podporované. I kdyby se jejich zpracování nějaký modul rozhodl sám implementovat, narazí na neznalost šablon. V podkapitole 3.5 je vysvětleno, že zanořené struktury vyžadují dohledání šablony pro jejich interpretaci až při samotném zpracování datového záznamu modulem. Šablony jsou ale ve správci šablon preprocesoru, se kterým už modul komunikovat nemůže.

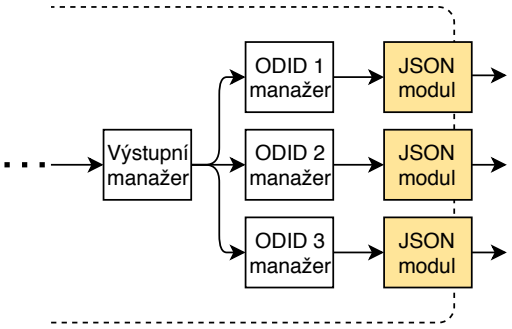
## 4.5 Konfigurace kolektoru a správa informačních elementů

Konfigurace kolektoru IPFIXcol se skládá ze tří souborů: přehledu instalovaných modulů, startovací konfigurace a seznamu definic informačních elementů. První jmenovaný konfigurační soubor určuje, kde v systému jsou moduly nainstalovány a jaké mají identifikační jméno. Na základě praktických zkušeností lze tvrdit, že tento soubor je možno zcela vypustit, jestliže by kolektor vyhledával moduly ve standardních cestách, kam se běžně ukládají knihovny. Útěchou současného stavu je fakt, že moduly jsou při své instalaci do konfigurace automaticky zahrnuty a uživatel nemusí soubor manuálně editovat.

Které moduly a s jakými parametry budou spuštěny, řeší **startovací konfigurace** kolektoru. Vzhledem k množství modulů, které mohou být současně spuštěny, a množství parametrů, které jsou u nich uživatelsky nastavitelné, by bylo značně nepohodlné vše zadávat jako parametry z příkazové řádky, a proto se používá konfigurační soubor. Při volbě struktury konfigurace se vycházelo z návrhu standardu RFC 6728 [13], jenž navrhuje unifikovaný způsob konfigurace napříč výrobci. Ten se avšak příliš neuchytil a na základě zpětné vazby od uživatelů, kteří mají problém se v konfiguraci vyznat, se ukázalo, že jeho volba nebyla příliš vhodná. Formát je značně strukturovaný a nepřehledný. Navíc návrh standardu nepodchycuje zdaleka všechny situace – vnitřní moduly vůbec neuvažuje. Bylo by vhodné konfiguraci značně upravit nebo alespoň vytvořit webový generátor konfigurací, kde by si uživatel zvolil jednotlivé moduly a jejich parametry.

Posledně jmenovaný konfigurační soubor souvisí s **definicemi informačních elementů**. Dovoluje uživatelům přidat nové elementy nebo stávající upravit. Nic není tvrdě zakódována jako v případě jiných kolektorů a není nutné s každou úpravou celý kolektor znovu zkompilovat. Každá definice obsahuje identifikátor (i s PEN), název, datový typ a sémantiku. Pro běžné interpretování záznamu je to postačující a kolektor navíc modulům nabízí rozhraní pro snadné hledání v těchto definicích. Nejsou zde ale podchyceny návaznosti na reverzní elementy a bez této znalosti biflow záznamy nelze identifikovat a ani interpretovat.

Schopností kolektoru je i **rekonfigurace** za běhu, což znamená možnost instance modulů vytvářet a ukončit dle potřeby, aniž by došlo k restartování programu. Pokud tak nechceme přijít o žádná data během potenciální odstávky, je možné pouze upravit konfigu-




Obrázek 4.3: Příklad duplikace samostatné instance výstupního modulu pro 3 různé ODID

raci a poslat běžícímu kolektoru signál. Kolektor si porovná startovací konfiguraci s předchozí verzí a provede příslušné změny. Na některé situace se ovšem nepamatovalo. Pokud instance modulu běží a chceme upravit její konfiguraci, kolektor změnu detekuje, instanci ukončí a spustí znovu, což obecně nelze považovat za správný postup. Někdy by se totiž hodilo, aby k ukončení nedošlo a instance si změny sama aplikovala. Například v případě výstupního modulu, který provádí distribuce IPFIX záznamů na další kolektory, vypnutí a zapnutí znamená ztrátu spojení se všemi klienty, což je nežádoucí. Při rekonfiguraci vnitřních modulů navíc dojde k celkovému zastavení zřetězené linky ihned za vstupním modulem, a dokud se nevyprázdní až po výstupní manažer, k rekonfiguraci nedojde. Hrozí tím pádem, že fronta za vstupním modulem bude zaplněna a vstup bude pozastaven.

Zásadně výraznější nedostatek má na svědomí způsob inicializace výstupních modulů. Původním záměrem kolektoru bylo co nejvíce zparalelizovat zpracování dat a to i při výstupu. Výsledkem bylo, že pro každé ODID vznikala samostatná kopie instance výstupních modulů. Pokud např. přicházela data ze 3 různých ODID, vznikly obvykle tři samostatné instance toho stejného modulu se stejnou konfigurací, což mohlo způsobit vzájemné kolize (např. snaha ukládat data do stejných souborů, poskytovat data jako server na stejném portu apod.). Konfigurace kolektoru sice dovolovala omezit daný výstupní modul pouze na jedno konkrétní ODID, ale to zase nedovolovalo zpracovávat a ukládat nebo dále distribuovat data z vybrané podmnožiny bodů monitorování v rámci jednoho souboru nebo proudu dat. Tento problém se později částečně vyřešil přidáním podpory pro jednotného výstupního správce dat, který vytváří pro všechna ODID jednu společnou instanci modulů, což ale na druhou stranu představuje druhý extrém, neboť omezuje dříve zamýšlené paralelní zpracování. Navíc stále přetrvává potíž, kdy inicializace instancí výstupních modulů je odložena až do příchodu prvních dat. Pokud je v konfiguraci chyba a inicializace se nepodaří, pokouší se výstupní manažer instanci znovu nastartovat s každou další příchozí zprávou, a tedy pouze mrhá výpočetními prostředky a generuje extrémní množství chybových hlášek.

## 4.6 Stav kódu, testovatelnost a dokumentace

Kvalita kódu různých částí kolektoru a modulů se značně liší v závislosti na schopnostech tehdejších autorů. Lze však pozorovat často opakující se **nežádoucí jevy**, které stojí za pozornost a je vhodné se jich v budoucnu vyvarovat. Jeden příklad za všechny. Kolektor nabízí modulům veřejné rozhraní pro práci s daty, a proto by funkce kolektoru měly být na první pohled snadno odlišitelné od funkcí, které si vytváří sám autor modulů. To ale neplatí, neboť neexistuje žádný společný prefix ani konvence, která by funkce spojovala.

Existují tak například veřejné funkce kolektoru pojmenované jako `tm_create_template`, `get_element_by_ids` nebo `data_record_get_field`. Při tvorbě a úpravách kódu modulů to může být velmi matoucí. Ani návrh řady komponent nezůstal bez poskvrny. Funkčností fronty pro předávání zpráv mezi instancemi modulů je i nečekaný vedlejší efekt. Je do ní zamíchán kód, který se při použití s výstupními moduly stará o zničení zaobalené zprávy, jakmile ji zpracovaly všechny instance, což mimo jiné zahrnuje uvolnění paměti a také dekrementace čítačů referencí použitých šablon. Takový neuvážený přístup značně narušuje testovatelnost a znovupoužitelnost kódu. Dále se vyskytují více či méně závažné stylistické prohřešky. Extrémní je ovšem situace u implementace vnitřní funkce `get_storage_plugins`, která ve svém těle obsahuje 10 úrovní zanoření programových bloků, z čehož 5 je součástí smyček. Nedosažitelné nebo nikde nepoužívané funkce už pak snad nikoho nepřekvapí.

Automatizované **testování kódu** je důležité pro udržitelný vývoj libovolného programu. V tomto případě existuje pouze pár základních systémových testů, které ověřují běh celého kolektoru v kombinaci s příjmem a výstupy vybraných modulů, a to ještě v dosti ideálních případech. *Testování jednotek* (angl. *Unit testing*) pro komponenty kolektoru zcela chybí. Přidání do stávajícího kódu by nebylo vůbec snadné, protože komponenty nebyly při svém návrhu a implementaci s tímto předpokladem stavěny. V kódech se ani nevyužívá kontrol pomocí příkazu `assert`. Jakákoliv úprava stávajícího kódu tak vede ke značnému riziku, že dojde ke skrytému rozbití kódu, a ztěžuje vývoj.

Pro dokumentaci kódu se používá *Doxygen*[3]. Většina funkcí je komentovaná i když poměrně plytce. Nejvíce je to znatelné na funkcích rozhraní kolektoru, kde řada otázek použití zůstává nezodpovězena a uživatel musí občas nahlédnout do implementace, což je špatně. Pro sestavení kolektoru ze zdrojových kódu se používá sada nástrojů známá pod názvem *Autotools*, která už je však poměrně stará, komplikovaná a dále nerozvíjená. Bylo by vhodné ji nahradit moderní alternativou.

[3] http://www.doxygen.org

# Kapitola 5

# Kolektor nové generace

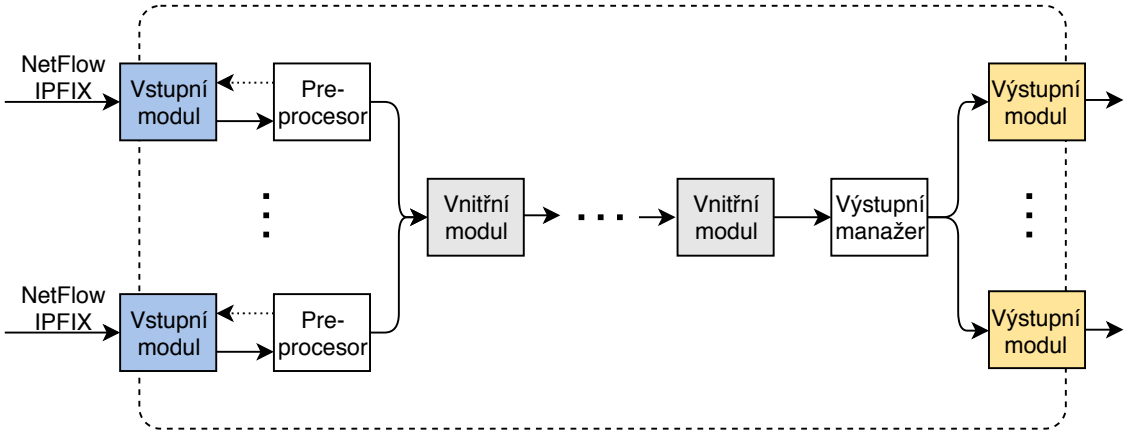
Předchozí kapitola poukázala na řadu ať už více či méně podstatných problémů v současném kolektoru. Mnoho nevalně navržených součástí, které jsou zakořeněné a vzájemně propojené, brání jakýmkoliv úpravám. Jediným východiskem je nová architektura a začátek s čistým štítem. Pro budoucí rozvoj je ale klíčové se ohlédnou na současné řešení a vyvodit z něj důsledky. Silné stránky návrhu zachovat, a ještě z nich více vytěžit. Na slabší stránky je třeba se více zaměřit a navrhnout nová řešení nebo vylepšení. Tato kapitola se zaměřuje na celkový návrh koncepce i jednotlivé součásti. Navrhuje nové přístupy, mírně upravuje koncept kolektoru a tím vytyčuje vývoj nejen v rámci této práce ale i základ prací budoucích.

Pro lepší pochopení řešení jednotlivých součástí kolektoru jsou v navazujících podkapitolách postupně diskutována a popisována řešení jednotlivých klíčových komponent nezbytných pro práci s NetFlow a IPFIX daty. U každé komponenty je poukázáno na současné řešení a jeho případné vylepšení, či přetvoření do podoby nové a více vyhovující praktickým potřebám.

## 5.1 Architektura jádra kolektoru

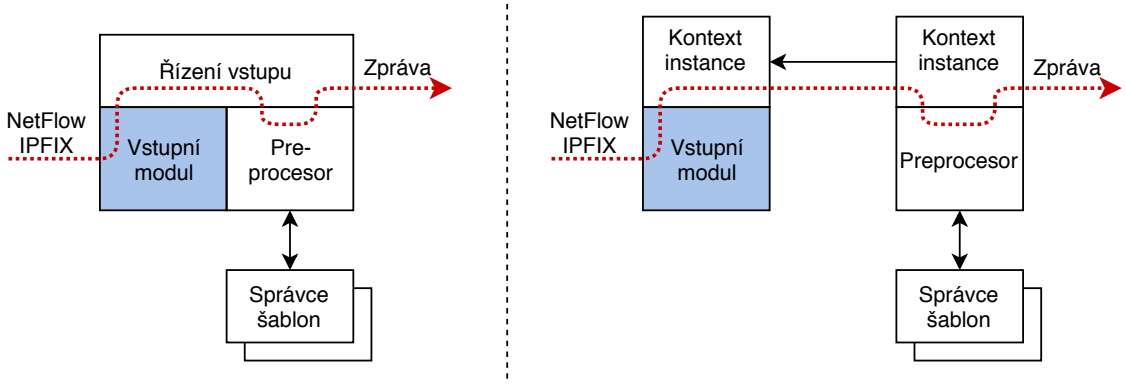
Samotná architektura dosavadního kolektoru z hlediska zřetězené linky je velmi dobrá. Nabízí paralelní zpracování a efektivní předávání zpráv mezi instancemi modulů. Koncept vstupních, vnitřních a výstupních modulů by tak měl být zachován, avšak zásadním úpravám se nevyhne vstupní a výstupní část linky. Nedostatkem předchozí architektury je, že dovolovala mít současně aktivní pouze jeden vstupní modul pro celý kolektor. Byť je sice možné zároveň spustit více samostatných kolektorů s odlišnou konfigurací, tyto kolektory mají zcela samostatné zřetězené linky, a není tak možné data přijímat skrze různé transportní protokoly a ukládat nebo je distribuovat společně v jednom proudu dat. Podpora **více vstupních modulů současně** ničemu neuškodí, a naopak dovolí rozložit zátěž na vstupu kolektoru na více paralelně běžících instancí různých nebo dokonce stejných vstupních modulů.

Další úprava spočívá v **oddělení** přímého napojení **vstupního modulu a preprocesoru**. Ve stávající architektuře řídí zpracování každé zprávy od vstupního modul až po výstup z preprocesoru jedno vlákno. To znamená, že vlákno musí nejprve surová data vyčíst ze síťového soketu, případně transformovat NetFlow do IPFIX, zpracovat zprávu, vytvořit zaobalující strukturu, vyřešit případné úpravy ve správě šablon a data poslat skrze frontu dalšímu modulu. Až je vše dokončeno, což může zabrat relativně dlouhou dobu, může teprve začít přijímat zprávu další. Poněkud skrytý problém ale nastává na systémovém rozhraní,




Obrázek 5.1: Architektura kolektoru nové generace

od kterého vstupní modul odebírá data. Pokud by systémový buffer pro síťové rozhraní byl relativně malý a vstupní modul neodebíral data s dostatečnou rychlostí, nastane nepříjemná situaci v okamžiku příchodu velkého nebo nárazového množství dat od exportérů. Při zaplněném síťovém bufferu buď bude docházet k zahazování zpráv ze strany systému (UDP, SCTP), nebo bude blokován exportní proces exportéru (TCP, SCTP). Nový návrh předpokládá samostatné vlákno obstarávající činnost instance vstupního modulu, který bude pouze surové zprávy ze sítě odebírat a posílat skrze nově vzniklou frontu k oddělenému preprocesoru, který také běží ve svém vlákně a bude provádět zbytek činností. Význam spočívá nejen ve snaze o zlepšení výkonosti, ale také v úsilí přesunou místo skrytého zahazování zpráv z rukou systému do místa vstupního modulu, který data může případně řízeně zahodit při zablokování vnitřní fronty. Preprocesor bude navíc schopen komunikovat i se vstupním modulem a posílat mu požadavky na ukončení spojení s exportéry, pokud se v datech vyskytne chyba. Srovnání původního a nového řešení ilustruje obrázek 5.2.




Obrázek 5.2: Detailní srovnání preprocesoru starého a nového kolektoru

Zásadním nedostatkem dosavadního kolektoru bylo ale počínání s výstupními moduly. Inicializace a jejích spuštění neprobíhalo při startu samotného kolektoru, ale bylo odloženo až do okamžiku, kdy byly přijaty první zprávy. Problematické zároveň bylo vytváření samostatných kopií instancí pro každé ODID zvlášť. Nový koncept výstupní mechanismus značně zjednodušuje tak, jak je vidět z porovnání obrázků 4.1 a 5.1. Výstupní manažer

nově již nebude vytvářet samostatné instance pro každé ODID, ale vždy vytvoří pro každou specifikaci výstupu **pouze jednu instanci** modulu, do které proudí ve výchozím stavu data ze všech ODID. Zároveň ale dovolí případně specifikovat filtry rozsahů ODID, které má výstupní instance zpracovávat a ostatní zprávy na ní nebudou z manažeru směrovány. V důsledku bude stále možné vytvářet více samostatných instancí stejných výstupních modulů ale s odlišnou konfigurací. Nad paralelizací bude mít uživatel mnohem větší kontrolu než doposud a každá výstupní instance bude moci zpracovávat vybrané intervaly ODID. Tento přístup s sebou nese navíc zásadní přednost v tom, že všechny výstupní instance je možné inicializovat již při startu kolektoru a v případě chybné konfigurace okamžitě přerušit start doposud nespuštěného kolektoru.

### Zřetězená linka

Zřetězená linka v nadsázce představuje jednosměrnou dálnici, po které cestují zprávy. V okamžiku, kdy samotné řešení linky není dostatečně efektivní, zpomaluje chování kolektoru jako celku. Stejně jako u svého předchůdce je zřetězená linka realizována skrze **kruhové buffery** jednosměrně propojující jednotlivé instance modulů. Nově navrhnutá architektura navíc vyžaduje takovou realizaci, která bude v případě více aktivních vstupních instancí současně umožňovat co možná nejefektivnější zápis z více instancí do jednoho bufferu zároveň. Za preprocesory totiž bude docházet ke spojování provozu do jednoho proudu jdoucího dále skrze vnitřní moduly až na výstup.

Součásti starající se o mazání zpracovaných zpráv nesmí být jako v předchozí verzi součásti bufferů. Tato činnost spíš více náleží řídícímu vláknu těchto instancí. Aby zároveň nemuselo na výstupu kolektoru docházet k výkonnostně náročnému kopírování zpráv na jednotlivé výstupní instance, je zpráva výstupními moduly sdílena. Pro realizaci je součástí každé hlavičky zprávy zřetězené linky i čítač referencí. Výstupní manažer si pouze určí na kolik instancí bude odkaz na zprávu, kterou dostal, přeposlán a podle toho jej nastaví a odkazy odešle. Jakmile je zpráva instancí zpracována, její řízení automaticky dekrementuje atomický čítač zprávy a pokud je nulový, zprávu bezpečně smaže.

Se zřetězenou linkou souvisí u dosavadního kolektoru jedna významná obtíž. Sloužila k předávání pouze jednotné struktury, a to konkrétně zaobalené zprávy od exportéru. Z pohledu vývojáře, který tvoří modul byl obsah struktury poměrně zmatečný, neboť současně nesl informace o stavu transportního spojení. Celá situace se ještě více zkomplikuje v okamžiku, kdy by struktura měla být obohacena o schopnosti nové, což by mohlo zapříčinit vznik dalších nekompatibilit. Moc potenciálu tento přístup do budoucna nenabízí a každá úprava jednotné struktury by vedla v lepším případě pouze k většímu zmatení uživatelů. Nový návrh tedy počítá s tím, že skrze zřetězenou linku bude moci chodit **více různých druhů zpráv**, kde každá zpráva bude ve své hlavičce obsahovat mimo jiné identifikátor příslušného typu. V tuto chvíli se počítá s druhy zpráv uvedených v tabulce 5.1.

Základ tvoří **datová zpráva** a **zpráva spojení**. První ze jmenovaných představuje absolutně nejčastěji vyskytující se zprávu s tím, že vždy obaluje surovou IPFIX zprávu, udržuje informace o původci dat, vyznačuje pozici záznamů ve zprávě a nese odkaz na jejich šablony. Zpráva spojení, jak už název napovídá, informuje o nově připojených exportérech nebo o uzavření stávajících spojení. Pokud to některý modul požaduje, může ji využít k uvolnění datových struktur navázaných na existenci spojení. Oba uvedené druhy zpráv jsou typicky generovány vstupními moduly.

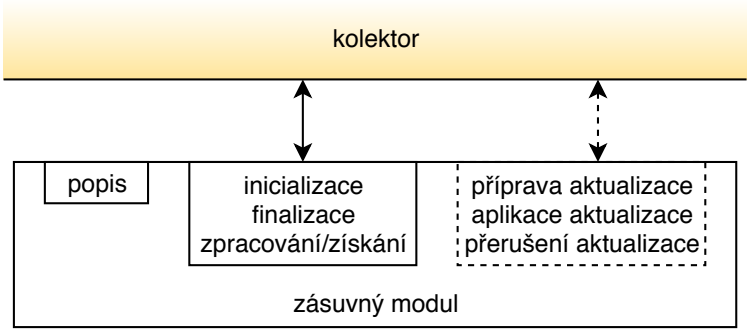
Některé zprávy jdoucí skrze linku jsou spíše servisního charakteru a není žádoucí, aby moduly s nimi jakkoliv manipulovaly. Mezi takové druhy zpráv patří konfigurační zpráva,

| Typ | Význam druhu zprávy |
|---|---|
| Zpráva datová | Zaobalující struktura nad IPFIX zprávou. Významově odpovídá struktuře použité v předešlé generaci kolektoru a to tím, že opět udržuje informace o pozici jednotlivých záznamů a provázání na šablony. Oproti předchozímu řešení už ale neobsahuje informace o stavu spojení a její implementace je skrytá skrze rozhraní. |
| Zpráva spojení | Informuje instance modulů o zahájení nebo ukončení transportního spojení s exportérem. Obsahuje identifikaci exportéru a způsob jeho spojení. Předchází vždy první datovou zprávu od daného zdroje a následuje za poslední. |
| Zpráva odpadní | Nese ukazatel na libovolnou datovou strukturu a funkci, která bude použita pro její uvolnění při destrukci odpadní zprávy. Využívá se pro zajištění pozdního uvolnění libovolné datové struktury v okamžiku, kdy by ji již nikdo ve zřetězené lince neměl používat. |
| Zpráva konfigurační | Používá se pro přípravu a aplikaci změn v konfiguraci kolektoru. Je využívána řídícím kódem instance volajícím odpovídající konfigurační funkce modulu. |
| Zpráva terminační | Slouží k bezpečnému ukončení kolektoru, když informuje řízení instancí modulů, že má dojít k jejich ukončení. V podstatě představuje absolutně poslední zprávu, která skrze zřetězenou linku prochází než bude definitivně zrušena. |

Tabulka 5.1: Druhy zpráv ve zřetězené lince

která je klíčová pro rekonfiguraci kolektoru, zpráva terminační, jež informuje o vypínání kolektoru, a zpráva odpadní, která nese již nepoužitelná data neznámého obsahu určená ke smazání. Tyto zprávy by měly být zpracovány pouze na úrovní řídících vláken jednotlivých instancí a neměly by vůbec být přímo zpřístupněny samotnému kódu modulu. S ohledem na možné budoucí rozšiřování typů zpráv je vhodné uvažovat, že každá instance si při své inicializaci zaregistruje druh zpráv, kterým rozumí a chce je přijímat.

Ze všech druhů zpráv zřetězené linky na první pohled vystupuje svojí atypičností **zpráva odpadní**, která nezapadá do běžné koncepce. Tento typ zprávy ovšem tvoří jednu z klíčových součástí nového návrhu. Nejlépe se její význam ilustruje na příkladu preprocesoru, který stejně jako v původní architektuře zpracovává příchozí IPFIX a NetFlow zprávy od exportéru. Ze zpráv preprocesor extrahuje šablony, které si uchová ve svém správci šablon, a každý přítomný datový záznam vyznačí a referencí spáruje s šablonou uloženou ve správci. Záznamy dále putují skrze zřetězenou linku a navazující moduly používají při interpretaci záznamů předchystané odkazy na šablony. Co se stane ale v okamžiku, když má být šablona v preprocesoru zničena, neboť se transportní spojení uzavřelo nebo byla šablona cíleně odebrána? Z principu není možné šablonu okamžitě zničit, protože kvůli paralelnímu zpracování mohou potenciálně stále existovat v kolektoru dosud plně nezpracované záznamy, které šablonu používají. Dosavadní kolektor tento rozpor řešil čítačem referencí. Nový kolektor ale staví na odpadních zprávách, které vychází ze základního předpokladu, že v rámci zřetězené linky se zprávy nikdy nepředbíhají a ani se neztrácí. Preprocesor jednoduše vygeneruje odpadní zprávu, do níž vloží již nepotřebnou šablonu, a pustí ji stejným způsobem jako zpracované datové zprávy skrze zřetězenou linku. Odpadní zpráva projde bez povšim-




Obrázek 5.3: Rozhraní zásuvného modulu

nutí skrze celou linku a v okamžiku, kdy je zpracována úplně všemi instancemi modulů její obsah, tj. šablonu, je možné bezpečně zničit. Jinými slovy platí, že všechny datové záznamy, které se nacházely v lince před touto zprávou a které si zároveň nesly odkaz na tuto šablonu, již musí být dávno zpracovány a zničeny.

### Zásuvné moduly

Zpovzdálí se na zásuvných modulech principiálně mnohé nezměnilo. Opět jsou realizované skrze dynamicky načítané knihovny, které implementují požadovanou funkcionalitu a s kolektorem komunikují skrze specifikované rozhraní. Při bližším pohledu ovšem na povrch vyplouvá celá řada více či méně významných detailů.

Nově je pro každou instanci modulu vytvořena **uniformní struktura kontextu**, která tvoří pomyslnou obálku každé instance a izoluje ji od ostatních. Kontext mimo jiné zahrnuje identifikaci instance, její privátní data, odkazy na funkce rozhraní implementované příslušným modulem, typy odebíraných zpráv a napojení na kruhové buffery. Podle typu modulu (vstupní, vnitřní, výstupní) běží nad kontextem řídící vlákno příslušného typu, které má na starosti přijímat data na vstupu kontextu a volat funkce náležící rozhraní modulu.

Paralelizace sice s sebou nese potenciál vyšší výkonnosti, ale na druhou stranu se stává nezřídka zdrojem dalších nesnází. Z pohledu kolektoru a jeho zřetězené linky je problematické používat globální komponentu, která by se s rekonfigurací měnila z pohledu všech instancí současně. Typickým příkladem je správce informačních elementů. Ve chvíli rázové rekonfigurace by mohl nastat typický problém nekonzistence, kdy část instancí vychází ještě ze staré konfigurace a jiné už využívají novou, což je mimochodem i jeden z nedostatků dřívější generace kolektoru. S tím souvisí současně otázka, kdy je možné dosavadního správce a jeho definice z kolektoru odebrat. Nově tuto obtíž řeší právě kontext instance, který si uchovává odkazy na systémové komponenty platné z jeho pohledu. V okamžiku, kdy dochází k rekonfiguraci a starý správce má být nahrazen novým, vyšle jádro kolektoru konfigurační zprávu, která projde celou zřetězenou linkou od počátku až po výstup a která postupně v jednotlivých kontextech nahradí staré odkazy na správce novými. Aktualizace tak tím dostává konzistentní řád a starého správce lze zrušit stejným způsobem jako šablony u preprocesoru, tj. s využitím odpadní zprávy.

Co se týče **rozhraní zásuvných modulů**, dojde ke změně jejích funkcí a předávaných parametrů. Zásuvné moduly by si měly ponechat tři základní dosavadní funkce, které musí implementovat, tj. inicializační, finalizační a procesní. Nově mohou volitelně podporovat i tři nové funkce pro dynamickou rekonfiguraci parametrů již běžících instancí. Jinak řečeno, pokud budou tyto nové funkce u nich implementovány a v konfiguraci dojde ke změně

jejich parametrů, instance nebudou restartovány, ale bude proveden pokus o změnu jejich parametrů za běhu. Pro stálost konfigurace kolektoru je důležité, aby se buď provedly všechny změny uvedené v konfiguraci, nebo nic. Je tedy nutné si změny prve připravit a až v okamžiku, kdy je vše připraveno, změny aplikovat nebo naopak zrušit. V případě, že se tvůrce modulu rozhodne tuto volitelnou funkcionalitu modulu implementovat, bude definovat navíc následující funkce.

- **příprava na aktualizaci** Na základě aktuální konfigurace a nových parametrů si instance modulu vytvoří dočasnou strukturu potřebnou pro budoucí změnu. Pokud je v nové konfiguraci chyba, může tuto situaci nahlásit a proces rekonfigurace bude přerušen.
- **aplikace aktualizace** V okamžiku, kdy všechny změny napříč moduly jsou již připraveny a nevyskytla se žádná chyba, je připravená struktura předána instanci k aplikaci změn.
- **přerušení aktualizace** Pokud byl proces rekonfigurace kýmkoliv přerušen a instance si připravila změny pro aktualizaci, je nutné připravené struktury aktualizace uvolnit.

Využití dynamické rekonfigurace lze nalézt tam, kde vypnutí a zapnutí výstupního modulu může vést ke ztrátě dat nebo spojení. Příkladem budiž výstupní modul přeposílající zprávy na další kolektory a snaha při rekonfiguraci přidat novou adresu nebo odebrat stávající adresu podřazeného kolektoru.

Knihovna modulu bude nově také obsahovat unifikovanou globální strukturu nesoucí název, typ a verzi modulu společně s minimální verzí kolektoru, se kterou je kompatibilní. Jádro kolektoru si pomocí této struktury snadno při načítání modulu ověří, o jaký modul se jedná. Pro budoucí potřeby by bylo vhodné vyhradit i prostor pro příznaky. Užitečným by se mohl stát např. příznak možnosti vytvořit pouze jedinou instanci tohoto modulu, což by přišlo vhod v okamžiku, kdy příslušný modul pracuje s externí knihovnou, která ovšem není připravena pro využití z více vláken (používá statické proměnné bez výlučného přístupu apod.).

### Konfigurace kolektoru

Konfigurace kolektoru se nově skládá pouze ze dvou částí, běhové konfigurace a konfigurace informačních elementů. Informační elementy jsou starostí příslušného správce a jejich bližší popis se nachází dále v této kapitole. Popis, jaké instance modulů mají v rámci kolektoru a s jakými parametry mají být spuštěny, má na starost právě **běhová konfigurace**. Byť by možná bylo ve vybraných případech snazší definovat spouštěné instance z příkazové řádky při startu kolektoru, snaha podporovat dynamickou rekonfiguraci tomu brání. Opět je tedy běhová konfigurace popsána skrze konfigurační XML soubor, avšak došlo ke značnému zpřehlednění. Nově konfigurace obsahuje tři části, které odpovídají jednotlivým typům zásuvných modulů, což je prezentováno na zkráceném výpisu kofigurace 5.1. Stále přitom platí, že musí být definována alespoň jedna instance vstupního a jedna instance výstupního modulu. Vnitřní instance mohou zcela chybět. Jediným výrazným rozdílem mezi těmito kategoriemi je skutečnost, že na pořadí specifikace instancí u vstupních a výstupních modulů nezáleží, neboť běží z pohledu architektury na stejné úrovni, což se ale o vnitřních instancích říci nedá a pořadí je pro konfiguraci kolektoru závazné.

```
<ipfixcol2>
    <inputPlugins>...</inputPlugins>
    <intermediatePlugins>...</intermediatePlugins>
    <outputPlugins>...</outputPlugins>
</ipfixcol2>
```

Výpis 5.1: Zkrácená konfigurace kolektoru

V rámci každé kategorie se nachází specifikace jednotlivých instancí. Ať už se jedná o vstupní (`<input>`), vnitřní (`<intermediate>`) nebo výstupní (`<output>`) instanci, pro každou z nich je povinně uvedeno označení instance, název zásuvného modulu a parametry, které budou předány při inicializaci. Ačkoliv se může jméno zdát nadbytečné, není tomu tak. Může-li být aktivních více instancí stejného modulu současně, musí být jádro kolektoru schopno při rekonfiguraci určit, u které instance se konfigurace změní, a právě jméno představuje unikátní identifikátor. Příklad zkrácené konfigurace jedné instance výstupního modulu je uveden ve výpisu 5.2.

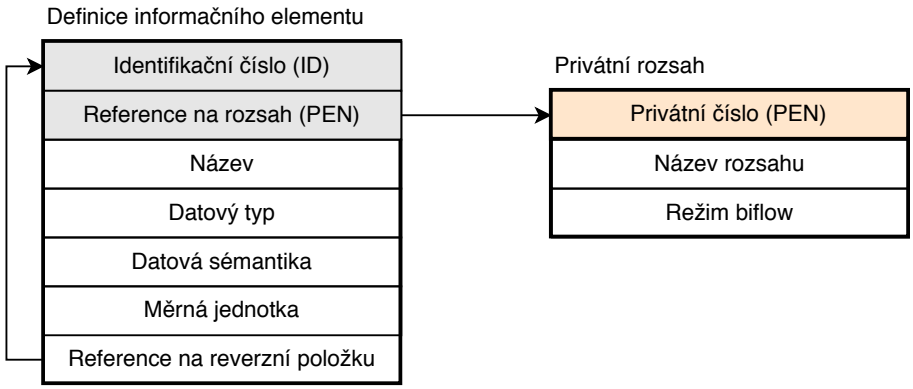
```
<outputPlugins>
    <output>
        <name>JSON storage</name>
        <plugin>json</plugin>
        <verbostity>debug</verbosity>
        <odidOnly>5-10</odidOnly>
        <params>
            ...
        </params>
    </output>
    ...
</outputPlugins>
```

Výpis 5.2: Zkrácený příklad konfigurace výstupního modulu

Při vývoji a používání modulů se ukázalo, že chybové a ladící výpisy jsou nezbytné pro odhalování provozních problémů. Globální konfigurace úrovně výpisů není vždy žádoucí, neboť při aktivaci ladících výpisů jsou generovány zprávy všemi instancemi, což snižuje přehlednost a komplikuje ladící aktivity. Nová konfigurace proto také dovoluje volitelně nastavit filtr úrovní výpisů pro každou jednotlivou instanci zvlášť. Z pohledu implementace to bude znamenat, že všechny moduly musí používat speciální funkce pro tvorbu zpráv svázané s jejich zaobalujícím kontextem. Detailnější pohled na konfiguraci je uveden v příloze C.

## 5.2 Informační elementy a jejich správa

V dřívějších kapitolách byl popsán význam informačních elementů jakožto základního stavebního prvku, který určuje a popisuje, jaké vlastnosti toků jsou měřeny a přenášeny prostřednictvím záznamů na kolektor. Mnoho nástrojů pracujících s NetFlow a IPFIX záznamy postrádá ucelenou správu informačních elementů a tuto problematiku řeší tím, že podporuje pouze nejdůležitější podmnožinu elementů, kterou má pevně zakódovanou ve zdrojovém kódu aplikace (například dříve zmíněný nástroj *nfdump*). Problém takového přístupu spočívá ve značně omezené flexibilitě aplikace, kdy uživatel není obvykle schopen přidat podporu pro zpracování dalších elementů, když např. pracuje s exportéry schopnými měřit nestandardní vlastnosti toků nebo experimentuje s novými rozšířeními exportérů.



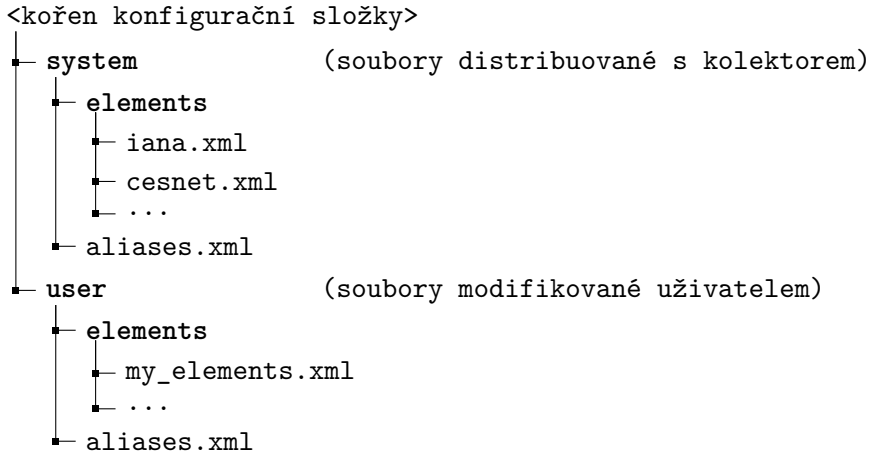

Obrázek 5.4: Informační element a jeho vztah s privátním rozsahem

Správce nejen, že musí řešit problematiku fixních definic, musí také poskytnou programové rozhraní pro samotné jádro kolektoru a zároveň pro jednotlivé rozšiřující moduly. Definice musí být dohledatelné minimálně na základě příslušné unikátní kombinace privátního čísla a identifikátoru, ale zároveň navíc musí skýtat vzájemné vztahy s jejich reverzními protějšky používaných v biflow záznamech. Obrázek 5.4 ilustruje vzájemnou návaznost mezi elementy a jejich definičním rozsahem.

### Souborová hierarchie

Dosavadní kolektor spravuje informační elementy tak, že je má definované v jednom strukturovaném XML souboru, který je distribuován společně s aplikací a je uživatelsky editovatelný. Byť by se mohlo zdát, že tento přístup řeší problém pevně definovaných elementů, je tomu tak pouze z části. Obecně se očekává, že kolektor po instalaci bude podporovat všechny běžné položky organizace IANA a dalších běžně rozšířených exportérů. Od uživatele se na druhou stranu očekává, že elementy navíc si sám případně definuje. V případě jednoho konfiguračního souboru, jak je provozováno v dosavadním případě, ale nastává problém při aktualizacích kolektoru, jenž zároveň nese průběžně aktualizovaný seznam definic. Nová aktualizace zcela nahradí seznam dosavadní a nechtěně odstraní uživatelem přidané definice. Jednou z možností by bylo při aktualizaci soubory porovnat, avšak tento přístup je poněkud těžkopádný, neboť by bylo nutné řešit případné kolize s uživatelskými definicemi. Nově navržený správce informačních elementů proto řeší tento problém tak, že předpokládá souborovou hierarchii uvedenou na obrázku 5.5.

Základem jsou 2 složky s naprosto totožnou vnitřní hierarchií, *system* a *user*. Zatímco složka *system* je spravována s aktualizacemi kolektoru, složka *user* není jimi nikterak dotčena a je určena pro uživatelsky modifikovanou část konfigurace. Obsah složek je dále strukturován tak, aby v budoucnu poskytnul prostor pro další případné konfigurační soubory – např. soubor *aliases.xml* s alternativními názvy informačních elementů pro jejich snazší identifikaci. Nový návrh dále předpokládá, že v rámci podsložky *elements*, se budou nacházet **samostatné XML soubory**, kde každý soubor zahrnuje definice elementů právě z jednoho rozsahu identifikovaného privátním číslem PEN. Správce informačních elementů, který s definicemi bude pracovat, prve načte systémovou konfiguraci a následně uživatelskou. V případě kolizí jsou systémové definice předefinovány uživatelskými, čímž se zajistí, že i v případě pozdější systémové aktualizace, která může přidat kolizní definice, dojde k zachování dosavadního označení uživatelských elementů a jejich parametrů.




Obrázek 5.5: Souborová hierarchie správce informačních elementů

### Struktura souboru s definicemi

Jelikož každý soubor představuje definice elementů z právě jednoho privátní rozsah, musí přirozeně obsahovat identifikační číslo PEN a také textové jméno pro lépe uchopitelné uživatelské označení. K tomu slouží XML element `<scope>` uvedený výpisu 5.3. Správce informačních elementů přitom dbá na to, aby číselné i textové označení bylo unikátní napříč všemi přítomnými rozsahy.

```
<ipfix-elements>
    <scope>
        <pen>0</pen>
        <name>iana</name>
        <biflow mode="pen">29305</biflow>
    </scope>
    ...
</ipfix-elements>
```

Výpis 5.3: Příklad definice rozsahu informačních elementů

Nový kolektor oproti předchozímu musí také podporovat práci s biflow záznamy a to bez přímé spolupráce správce nelze. Jelikož takovéto datové záznamy se skládají z běžných elementů (můžeme je pro tyto účely také označit za dopředné) a elementů reverzních, které vyjadřují opačný směr, musí být kolektor schopen směr jednoznačně určit. Na reverzní elementy lze pohlížet jako na zrcadlení svých dopředných protějšků, přičemž sdílí naprostou většinu svých parametrů a liší se pouze v identifikačním čísle a pojmenování. Nasnadě je tedy vyhnout se potenciální chybné duplikaci položek.

Návrh standardu popisujícího biflow toky [17] zavedl v případě společného rozsahu (PEN 0) vytvoření odděleného rozsahu (PEN 29305), který doslova zrcadlí dopředné elementy. Kolektor, který narazí v záznamu na položku s tímto rozsahem může snadno dospět k poznání, že se jedná o reverzní element. Situace ovšem není tak přímočará u privátních rozsahů, tj. nestandardních informačních elementů, kde neexistuje jeden unifikovaný přístup a každý správce daného rozsahu si může reverzní položky označit jiným způsobem. Z tohoto důvodu konfigurace počítá se čtyři druhy biflow režimů v rámci konfigurace:

**pen** Vytvoření odděleného rozsahu s definovaným identifikačním číslem, ve kterém jsou všechny položky zrcadleny oproti stávajícímu seznamu definic. Tento princip je použit v případě společných elementů IANA.

**split** Rozsah, jenž může nabývat až $2^{15}$ definic je rozdělen typicky na dvě poloviny, kde dolní polovina (ID 0 - 16383) představuje elementy dopředné a horní poloviny (ID 16384-32767) zrcadlené elementy reverzní. Takovýto přístup je doporučený návrhem standardu pro privátní rozsahy.

**individual** Pouze vybrané elementy mají své reverzní protějšky a ty to přímo explicitně uvádí.

**none** reverzní informační elementy nejsou v daném rozsahu podporovány.

Správce elementů si při načítání konfigurace automaticky odvodí na základě použitého režimu definice reverzních elementů, čímž pádem stačí, že se v konfiguračním souboru nalézají pouze elementy dopředné.

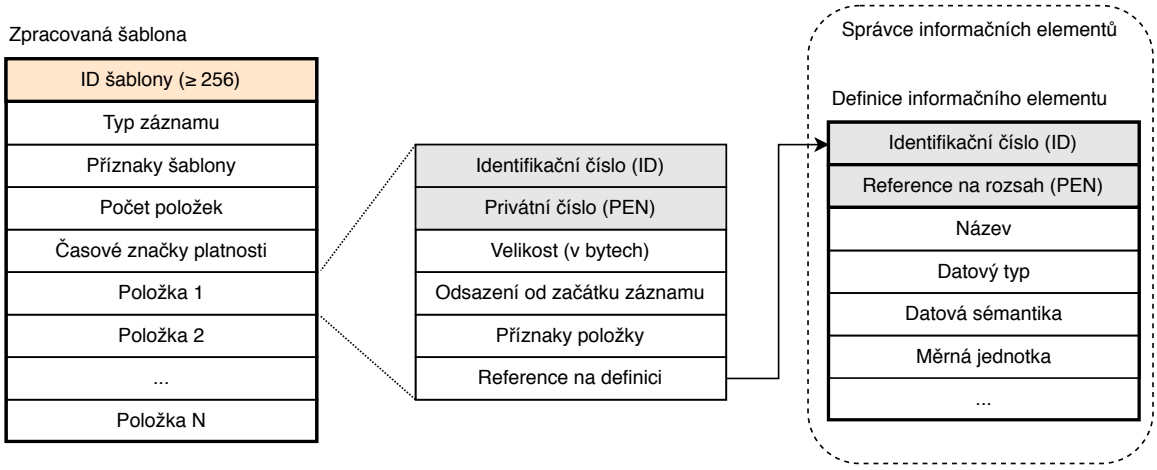
V rámci vymezeného rozsahu se vyskytují obvykle definice více informačních elementů, které se skládají z identifikátoru, názvu, datového typu, sémantiky, měrných jednotek a stavu platnosti. Mezi povinné se řadí pouze první tři vlastnosti. Pokud rozsah, do kterého náleží, využívá individuální režim konfigurace reverzních elementů, může definice navíc obsahovat `<biflowId>` vyjadřující identifikátor zastupující odpovídající reverzní prvek ve stejném rozsahu.

```
<ipfix-elements>
    ...
    <element>
        <id>1</id>
        <name>octetDeltaCount</name>
        <dataType>unsigned64</dataType>
        <dataSemantics>deltaCounter</dataSemantics>
        <units>octets</units>
        <status>current</status>
    </element>
    ...
</ipfix-elements>
```

Výpis 5.4: Příklad definice informačního elementu

## 5.3 Šablony a jejich správa

Jak již bylo mnohokrát zmíněno, exportéry mohou u různých toků zachytit různé vlastnosti a tyto různorodé záznamy jsou následně přenášeny na kolektor. Aby byl kolektor schopen rozpoznat, jaké vlastnosti jsou v binárních záznamech toků přítomny a kde přesně se nacházejí, používají se šablony, které odpovídající záznamy popisují. Protože exportér obvykle současně posílá více druhů záznamů (např. minimálně kvůli odlišnému druhu IP adres v záznamech) musí si kolektor pro každý exportér spravovat jeho šablony záznamů, aby mohl jeho data interpretovat. Tato různorodost s sebou na jednu stranu dává záznamu datovou flexibilitu, která na druhou stranu komplikuje samotné zpracování a manipulaci s daty. Rozšíření přijaté šablony na straně kolektoru o další parametry potřebné pro usnadnění práce proto dává smysl.




Obrázek 5.6: Struktura zpracované šablony a návaznost na informační elementy

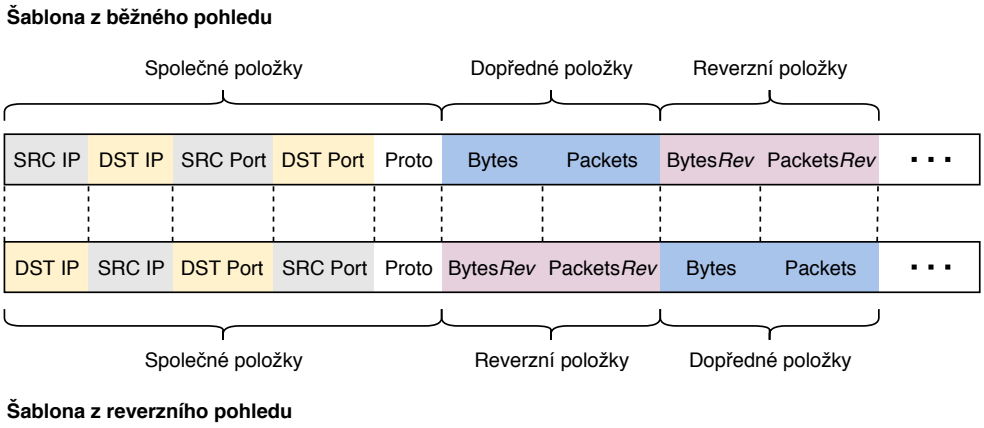
### Zpracovaná šablona

Běžná šablona exportérem zaslaná kolektoru obsahuje jen naprostý základ nezbytný pro rozlišení jednotlivých položek v záznamu. Lze z ní určit pouze identifikační čísla informačních elementů, pořadí a velikost jednotlivých položek záznamu. Dosavadní kolektor si každou šablonu pouze mírně rozšířil o přepočítané položky, jako je celková velikost (statického) záznamu a odsazení položek nejčastěji používaných informačních elementů, tj. adres, portů a počtu bytů a paketů.

Jak se ale ukazuje, pokročilejší práce se záznamem vyžaduje mnohem **více odvozených rysů**, které jsou mnohdy získatelné jen v návaznosti na znalost přítomných informačních elementů. V původním kolektoru byla správa informačních elementů přidána až v pozdějším vývojovém stádiu, což zapříčinilo, že zpracované šablony nejsou nijak navázány na definice informačních elementů. To má osudný dopad například u konverze datových záznamů výstupními moduly na textovou podobu, neboť při konverzi se pro každou jednotlivou položku ve správci pracně dohledává odpovídající definice informačního elementu nesoucí datový typ nutný pro to, aby se položka správně transformovala. Při vysokém vytížení kolektoru lze předpokládat, že může docházet ke konverzi více jak stovek tisíc záznamů za sekundu, kdy neustálé dohledávání jednotlivých definic už má citelný výkonnostní dopad.

Nová generace kolektoru již tyto problémy řeší. Dříve představený pokročilý správce informačních elementů je již součástí od samotného počátku návrhu a zpracovaná šablona návaznost na definice patřičně využívá. Obrázek 5.6 jasně ilustruje, že každá položka zpracované šablony přináší přímé odkazy na definice přítomné ve správci, odsazení od počátku záznamu (nenachází-li se před ní žádná položka dynamické délky) a zároveň si nese bitové příznaky následujících vlastností:

- **vícenásobný výskyt** Pomocná informace vyjadřující, že položka reprezentující tentýž informační element je v šabloně přítomna vícekrát. Může sloužit při filtrování v záznamů, kdy se hledá alespoň jeden odpovídající výskyt vyhovující vzoru.
- **poslední výskyt** V návaznosti na předchozí příznak vyjadřuje informaci, že je již žádná další taková položka se dále v záznamu nenachází a slouží de facto jako zarážka prohledávání.




Obrázek 5.7: Srovnání šablony z dopředného a reverzního pohledu

- **rámcová položka** Přítomná pouze u šablon záznamů nastavení a provozních parametrů (angl. *Options Template*) označuje, že položka je součástí definice rámce, či kontextu, ke kterému se vztahují data záznamu.
- **strukturovaná položka** Jedná se jeden ze tří strukturovaných datových typů, který je blíže určen u definice informačního elementu.
- **reverzní položka** Informační element příslušné položky záznamu náleží reverznímu směru u biflow záznamů.
- **společná položka** U biflow záznamů označuje informační element náležící položce sdílené oběma směry záznamu.

Je záhodno si všimnou, že poslední tři vlastnosti není možné určit bez přítomnosti a bez napojení na definice odpovídajících elementů. Zejména pro určení šablon biflow záznamů jsou definice naprosto klíčové, neboť by bez nich nebylo ani možné stanovit, zda se jedná o tento typ záznamu, a právě u biflow šablon se na chvíli zastavíme. Každý jejich záznam se skládá z položek běžných informačních elementů, které pro snazší identifikaci můžeme označit za dopředné, a elementů reverzních.

K určení, zda se jedná o **biflow šablonu** stačí nalezení alespoň přítomnosti jedné reverzní položky v šabloně. V tem okamžik je nevyhnutelné šablonu dále hlouběji analyzovat. Nejprve se určí, které vlastnosti jsou sdílené a které náleží pouze jednomu směru komunikace. To lze poměrně jednoduše. V záznamu se identifikují všechny položky reverzních informačních elementů a na základě nich se dohledají v tom samém záznamu položky patřící odpovídajícím dopředným informačním elementům. Ostatní položky šablony, které obvykle patří klíčovým vlastnostem toku, lze chápat jako společné pro oba směry a jsou vyjádřeny z dopředného pohledu. U těchto položek navíc rozlišujeme zdrojové, resp. cílové položky, které jsou se svých oficiálních textových označeních uvozeny slovy „source“ (např. zdrojová IP adresa nebo zdrojový port), resp. „destination“ (např. cílová adresa nebo cílový port). Společné položky, jenž tuto podmínku nesplňují (např. typ transportního protokolu), se označují jako nesměrové.

Tohle na první pohled nepodstatné rozlišování vlastností má ovšem zásadní význam. Je třeba se uvědomit, že na biflow šablonu lze nahlížet nejen z běžného směru, ale i ze směru reverzního. Uplatnění lze najít v případě, kdy potřebujeme rozložit biflow záznam na dva

samostatné jednosměrné záznamy. To, že na šablonu lze hledět z obou směrů, znamená, že zpracovaná šablona v novém návrhu kolektoru si pro biflow případ vytvoří druhou kopii zpracovaných elementů, kde jsou nesdílené dopředné identifikátory položek nahrazeny za odpovídající reverzní a naopak. U směrových sdílených položek dojde k přeznačení položek zdrojových elementů na cílové a naopak. Obrázek 5.7 ilustruje, jak takto zpracovaná šablona vypadá.

Nesmíme zároveň zapomínat, že ne všechny šablony popisují záznamy toků. Některé reprezentují i záznam nastavení a provozních parametrů na straně exportéru. Kdokoliv, kdo by s takovým záznamem cíleně pracoval, by si musel pokaždé určit, co vyjadřuje podle přítomnosti informačních elementů. Například standard RFC 7011 [4] udává, že existuje záznam tzv. *statistiky spolehlivosti exportního procesu* (angl. *Exporting Process Reliability Statistics*), který v sobě musí minimálně obsahovat položky informačních elementů popisující identifikaci exportéru, začátek a konec sledovaného období a množství zahozených záznamů toků, které se nepodařilo na kolektor odeslat. Neustálé dohledávání položek by bylo kontraproduktivní, a proto právě takové typické a standardní identifikace budou detekovány nově při prvotním zpracování šablony.

Kromě příznaků, které náleží jednotlivým položkám šablony, lze navíc rozlišovat i příznaky náležící šabloně jako celku a ty mohou značně usnadnit nebo jinak urychlit zpracování záznamů dle dané šablony:

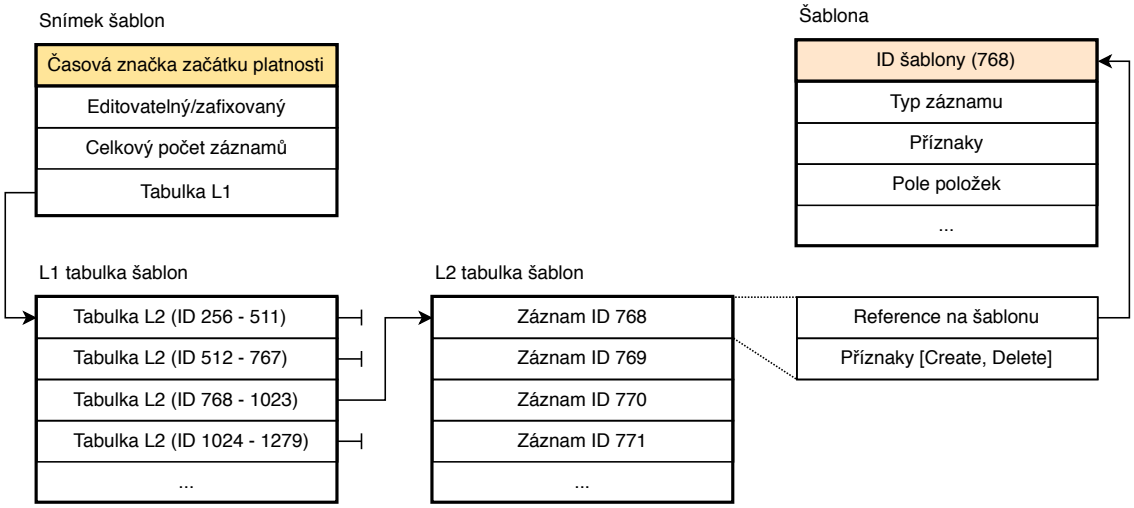
- **vícenásobný výskyt** Vyjadřuje skutečnost, že v šabloně se nachází alespoň jedna položka náležící stejnému informačnímu elementu vícekrát.
- **dynamický záznam** Šablona obsahuje alespoň jednu položku dynamické velikosti, jejíž velikost je až součástí samotného záznamu, a tím pádem nelze použít předpočítanou celkovou velikost k určení hranic záznamu.
- **biflow záznam** Příslušný záznam popisuje obousměrný tok a příznak zároveň signalizuje, že součástí šablony je i kopie zpracovaných položek z reverzního pohledu.
- **strukturovaný záznam** Součástí záznamu je alespoň jedna dále strukturované položka, což například při filtraci znamená, že je nutné záznam procházet i do hloubky.

### Správce šablon

Pouze minoritní část zpráv od exportéru přicházející na kolektor obsahuje definice šablon. Zbytek zpráv, které po nich následují a které nesou datové záznamy, se na ně pouze odkazují. Nejenže šablony mohou přibývat, ale mohou být v průběhu komunikace také rušeny. Správa šablon a schopnost kolektoru v nich dohledávat je tedy naprosto elementární nutností. Dopředu je také třeba si stanovit některé požadavky, ujasnit si problémy, které mohou nastat, a v neposlední řadě je v novém návrhu zapotřebí se vyvarovat chyb předchozí generace.

Úplně nejzákladnějším kritériem nutným v případě paralelního zpracování dat je garance, že příchod nových šablon nebo jejich rušení na začátku zřetězené linky se rázem neprojeví na konci linky, kde se ještě potenciálně zpracovávají starší záznamy. Na jednou zpracované šablony tudíž musí být následně pohlíženo jako na neměnné struktury. Dalším cílem nového kolektoru bylo zajištění podpory pro strukturované datové typy, jenž předpokládají, že při zpracování záznamů, v nichž se nachází, je dostupná nejen samotná hlavní šablona tohoto záznamu, ale zároveň je možné dohledávat všechny šablony platné v okamžik příchodu záznamu na kolektor. Jinak řečeno nesmí být dohledatelné šablony starší a již neplatné nebo

naopak novější, které byly definovány až po samotném záznamu. V kapitole analyzující starý kolektor bylo tehdejšímu správci vyčítáno, že zcela rezignoval na práci s exportními časovými značkami záznamů a neumožnil dohledávat v nutných případech šablony v historii pro opožděné záznamy. Všechny tyto problémy musí být v návrhu adresovány.




Obrázek 5.8: Snímek šablon a dvouúrovňová adresovací tabulka

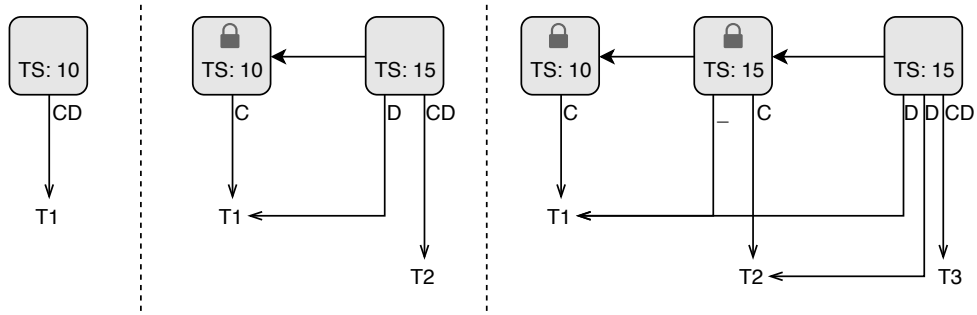
Nejprve se zaměřme na samotný způsob, jak by bylo možné šablony uchovávat v jednom okamžiku. Ignorujme na chvíli paralelismus kolektoru a představme si, že všechny činnosti od příjmu záznamu až po výstup probíhají sekvenčně tj. než by vstupem kolektoru byla přijata nová zpráva, předchozí by musela být plně zpracována. Počet existujících referencí na šablonu by nebylo nutné vůbec řešit a šablony šlo velice snadno přidávat a rušit. Bylo by možné se inspirovat dosavadním řešením, kdy si správce uchovával tabulku s náhodně uspořádanými referencemi, a trochu ho vylepšit tak, že reference by byly setříděny a bylo by možné aplikovat binární vyhledávání. Přidávání a odebírání šablon, byť neprobíhá velmi často, by vyžadovalo pokaždé setřídění. Nabízí se ovšem i jiné řešení, na kterém nový správce staví. Pokud totiž přihlédneme k limitovanému množství šablon, které mohou současně existovat tj. 65 280 (dáno rozsahem ID 256 – 65535), lze uplatnit mnohem přímočařejším přístupu a to v podobě **dvouúrovňové adresovací tabulky**, jak je uvedeno na obrázku 5.8. Vkládání a odebírání šablon nevyžaduje setřídění a dohledání probíhá v konstantním čase. Druhé úrovně tabulky přitom nemusí existovat, pokud nejsou využity a lze je případně vytvářet až v potřebný okamžik.

Komplikaci nám ovšem vytváří paralelismus kolektoru a požadavek na schopnost uchovat si historii platných šablon potřebných pro interpretaci strukturovaných typů. Jednou z možností by bylo, aby preprocesor zpráv na vstupu kolektoru pro každý jednotlivý záznam a každou jeho strukturovanou položku vyžadující odkaz, připravil odkaz na potřebnou šablonu. Samotné struktury mohou být ovšem dále strukturované a musel by tak činit pro každou úroveň, což by pravděpodobně vedlo k celé řadě dynamicky alokovaných drobných struktur s nepříznivým dopadem na výkonost. Nový návrh k tomu přistupuje jinak. Správce šablon bude podporovat **vytváření snímků** (angl. *snapshot*) všech šablon platných v okamžiku zpracování datové sady náležícího záznamu. Odkaz na tento snímek (určený výhradně pro čtení) bude součástí každého zpracovávaného záznamu. Kdokoliv, kdo bude chtít pracovat se strukturovanými typy, si pak snadno sám dohledá odpovídající šablonu ve

snímku. Zároveň se ale není třeba ani obávat zvýšené režie, protože nové snímky se budou vytvářet jen a pouze v okamžiku změny šablon.

Snímek je realizován tak, jak je uvedeno na obrázku 5.8 s využitím 2 úrovňové vyhledávácí tabulky, a přitom obsahuje pouze informaci o exportní časové značce, režimu editovatelnosti a pro každou šablonu si navíc uchovává speciální příznaky.

- **Příznak tvorby (create)** Snímek je úplně první, kde se reference na tuto konkrétní šablonu kdy vyskytla. Jedná se pouze o pomocný příznak sloužící jako zarážka, při prohledávání historie výskytu dané šablony mezi snímky. Při duplikování snímků se příznak nikdy nepředává.
- **Příznak smazání (delete)** Označuje poslední, tj. nejčerstvější, snímek šablony, který referenci na danou šablonu obsahuje a zároveň mu dává zodpovědnost zkontrolovat při svém rušení, zda je posledním platným snímkem, kdo šablonu používá, a případně jí musí také smazat. Při duplikování snímků se příznak přenáší na novější.

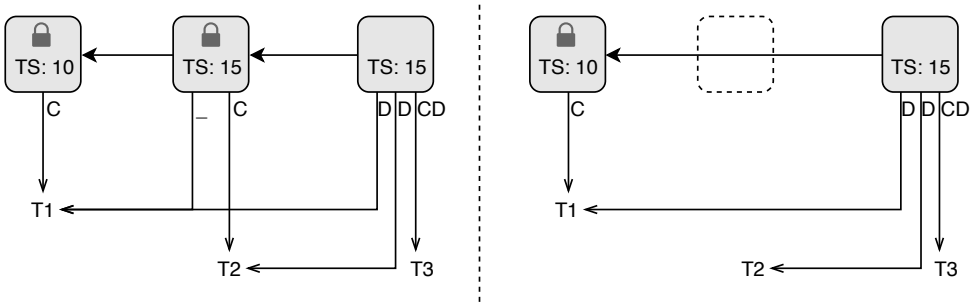



Obrázek 5.9: Podoba snímků správce šablon při postupném přidávání šablon

Nový správce šablon bude pracovat se snímky následujícím způsobem, který je i současně prezentován na obrázku 5.9. Na počátku startu komunikace s exportérem se předpokládá, že žádné předem určené šablony neexistují, a tím pádem neexistuje ani žádný snímek. V okamžiku příchodu první definice šablony se vytvoří snímek s exportním časem zprávy, ve kterém byla definována. Protože je snímkem, který šablonu vytvořil, má u této reference veden příznak tvorby. Současně je v danou chvíli také posledním snímkem, který na šablonu odkazuje, a tak mu náleží i příznak smazání. Jakmile nastane situace, že přijde datový záznam, který požaduje odkaz na snímek, dojde k zablokování editace tohoto snímku a žádné šablony již nemohou být přidány nebo odebrány. Dovolena je správci pouze změna příznaků šablon, neboť ta z pohledu dohledávání odkazů na šablony nehraje roli. Při zpracování dalších definic šablon (přidání či odebrání) se kontroluje, zda aktuální snímek je editovatelný a má stále stejný exportní čas záznamu. Pokud tomu tak není, vytvoří se s patřičnou exportní časovou značkou duplikát snímku, který obsahuje odkazy na všechny šablony, přičemž dojde k přenosu všech příznaků smazání, které se ve zdrojovém snímku nacházely na snímek nový, ale příznaky tvorby se nepřenáší. Na použitém principu je zejména důležité si všimnout, že samotné šablony se nikdy nekopírují a nepoužívají se žádné čítače referencí na šablony ani na snímky.

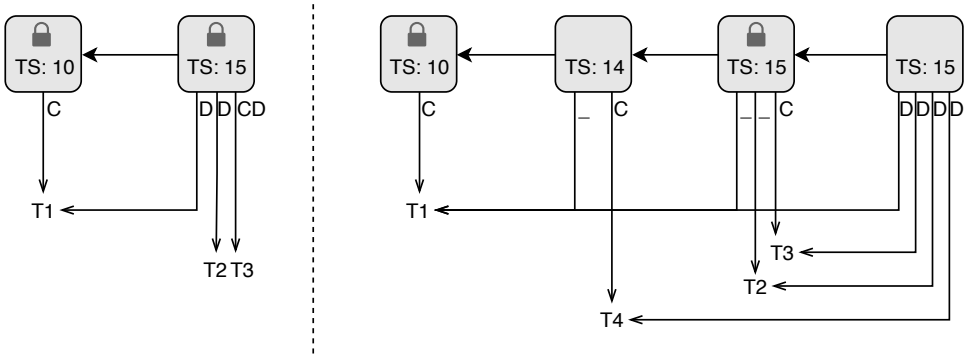
Správce šablon si snímky udržuje seřazené v lineárně zřetězeném seznamu dle exportních časových značek od nejstaršího po nejnovější. **Rozsah časové platnosti snímku** je z leva ohraničen jeho časovou značkou a zprava značkou jeho nástupce. Pokud přijdou opožděné

záznamy, tj. záznamy obsažené ve zprávě se starším exportním časem než je poslední známý, je nutné dohledat šablony pro jejich interpretaci a také odpovídající snímek. V ten okamžik správce snímky prochází ze směru od nejnovějších k nejstarším a hledá první snímek, do jehož intervalu platnosti spadá exportní čas zprávy. Pokud např. existují dva snímky s časem TS:10 a TS:15, tak první snímek náleží všem nově příchozím (opožděným) záznamům s exportním časem 10 – 14, včetně.




Obrázek 5.10: Správce šablon před a po odebrání nedosažitelného snímku

Na obrázku 5.10 si lze všimnout situace, kdy byly vytvořeny dva snímky se stejný exportním časem. To se mohlo například stát tak, že u původního snímku došlo k zablokování editace ve chvíli, kdy alespoň jeden datový záznam na něj získal odkaz. Následně přišla v rámci stále stejného exportního času další šablona, a protože snímek již nebyl editovatelný, došlo k vytvoření jeho duplikátu, do něhož se nová šablona zařadila. Starý snímek byl doslova překryt novějším a žádným nově příchozím záznamům již nemůže být přiřazen. Takový snímek stejně jako snímky sahající relativně hluboko do historie je možné již ze správce vypustit. Při odebírání snímků je ale nutné dbát na to, že mohou být posledními, kdo si uchovává referenci na některou ze svých šablon a musí jí odebrat také. K tomu účelu existuje příznak smazání, který u referencí může být nastaven. Pokud se tedy snímek odebírá, správce projde všechny jeho šablony s příznakem smazání a pro každou se směrem zpět do historie pokusí najít první další platný snímek, který na šablonu odkazuje. Pokud se mu to podaří, znamená to, že stále existuje ještě jiný snímek, jenž na šablonu odkazuje a není ji možné odebrat. Takovému snímku je předána zodpovědnost za odstranění oné šablony přenesením příznaku. Je vhodné zmínit, že shodným způsobem jsou odebírány jednotlivé šablony z aktuálního editovaného snímku v okamžiku, kdy na žádost exportéru je nutné je zneplatnit nebo jejich platnost vypršela po uplynutí časového limitu.




Obrázek 5.11: Správce šablon před a po přidání opožděné šablony

Pozornému oku neunikne fakt, že nejen záznamy mohou chodit opožděně, teoreticky může přijít opožděně i definice šablony a s tím je nutné ve správci rovněž počítat. Obrázek 5.11 jasně ilustruje takovou situaci. V tomto konkrétním případě došlo k opožděnému příchodu definice šablony s exportním časem zprávy TS:14. Správce šablon si našel v historii snímek, do jehož rozsahu exportní čas zapadá, vytvořil jeho duplikát s odpovídající časovou značkou a přidal do něj šablonu. Poněvadž se nově vytvořený snímek nachází v historii, musí dojít k propagaci změn. Pokud novější snímky jsou stále editovatelné, mohou se změny provést přímo v nich. Pokud tomu tak není, musí se vytvořit jejich duplikáty, jež budou změny reflektovat.

Aby všech komplikací nebylo málo, správce u šablon přenesených skrze transportní protokol UDP řeší **životnost šablon**, kdy po uplynutí stanoveného časového limitu už nejsou dále platné, pokud nedošlo k jejich redefinici. To z praktického pohledu znamená, že až zprávy od exportéru dosáhnout exportního času, kdy již šablona více neplatí, správce automaticky generuje snímky nové bez referencí na neplatnou šablonu.

Z nového návrhu lze jasně vidět, že správce je v mnohých ohledech značně pokročilejší vzhledem ke svému protějšku z dřívější generace kolektoru. Nejenže řeší všechny jeho nedostatky, zakládá navíc na principu snímků šablon platných v okamžik příchodu jednotlivých zpráv. V hlubším důsledku tím zajišťuje konzistentní pohled na šablony záznamů při zpracování zřetězenou linkou na jejím začátku i konci.

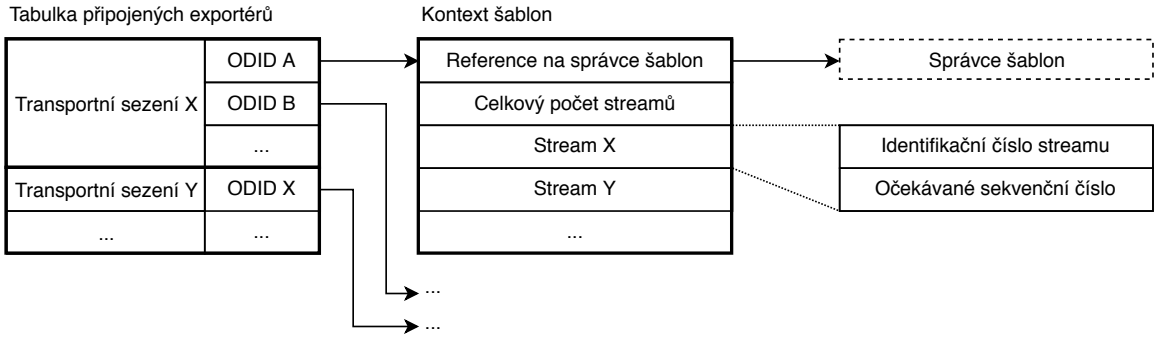
## 5.4 Zpracování zpráv a práce s datovými záznamy

Zřetězená linka, zásuvné moduly, informační elementy a šablony. Všechny dosavadní součásti vedly k jednomu společnému cíli a tím je zpracování datových záznamů od exportéru. Bez informačních elementů by nebylo možné jednoduše identifikovat, jaké vlastnosti jsou v šablonách záznamů obsaženy. Bez šablon by nebylo zhola možné interpretovat obsah jednotlivých záznamů. Než jsou však záznamy z IPFIX a NetFlow zpráv připraveny pro zpracování, musí dojít k předzpracování preprocesorem.

### Preprocesor

Již z pohledu zařazení do zřetězené linky se osamostatnil od vstupního modulu. Jeho vnitřní chování se ale také musí dočkat změny. Není radno podceňovat preprocesor z hlediska vykonávaných činností, a proto je vhodné jeho konání postupně rozebrat. Základem pro jeho činnost jsou surové zprávy převzaté od instance vstupního modulu. Tyto zprávy ovšem mohou chodit skrze transportní sezení od různých exportérů a z různých míst měření sítě. Jak ale zajistit, aby preprocesor měl vždy povědomí, od koho data pochází, když sám s nimi přímo neinteraguje a zprávy dostává skrze zřetězenou linku? Návrh nově počítá s tím, že samotná `zaobalující datová zpráva` zřetězené linky bude vytvořena již vstupním modulem, který do ní neoddělitelně vloží informace o původci dat, a pošle ji preprocesoru. Ten ji přijme, případně provede konverzi z NetFlow na IPFIX, projde skrze surovou zprávu, kde vyextrahuje šablony a v zaobalující zprávě vyznačí přítomnost záznamů. Na závěr předá zprávu další instanci ve zřetězené lince. Navíc kvůli podpoře strukturovaných datových typů vyznačené záznamy již nově nemohou být identifikovány pouhou trojicí (začátek záznamu, délka a šablona), ale musí dojít k rozšíření na čtveřici obsahující navíc i odpovídající snímek šablon.

Čítače referencí šablon jsou nyní aktivně nahrazovány za odpadní zprávy. Pokud tak má dojít ke zrušení šablony nebo snímku, bude se pro ně generovat odpadní zpráva, která

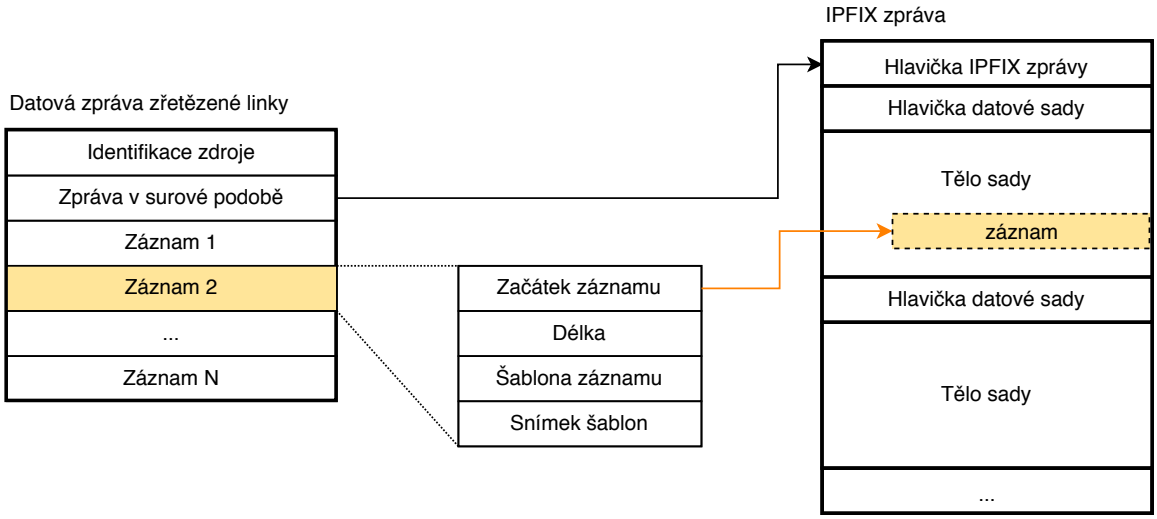



Obrázek 5.12: Vnitřní datové struktury preprocesoru pro transportní sezení a ODID

bude vložena do zřetězené fronty za datovou zprávu, která jejich zrušení způsobila. Tento přístup je naprosto korektní, protože ve zřetězené lince zprávy nemohou měnit pořadí, a tak poslední datové záznamy, které na ně mohou ukazovat, budou vždy po proudu linky dále. Až dojde na rušení odpadní zprávy a mazání, v tu chvíli už všechny záznamy, které ji kdy používaly, nebudou existovat. Stejný princip je využíván i v případě ukončení spojení s exportérem, kde obsahem odpadní zprávy je celý správce šablon.

Nezapomínejme, že šablony jsou vždy platné pouze v rámci unikátní kombinace transportního spojení a ODID. Pro tyto účely si nový preprocesor průběžně vytváří spojení tak, jak je reprezentováno obrázkem 5.12. Pro každou kombinaci si uchovává zcela nezávislého správce šablon, do kterého vkládá zpracované šablony, na právě které vyznačené datové záznamy odkazují. Vedlo toho si zároveň vede informace o předpokládaném sekvenčním čísle následující zprávy, aby dokázal určit, zda případně nedošlo při přenosu od exportéru ke ztrátě. Drobnou komplikaci zde zanáší transportní protokol SCTP, který umožňuje přenášet zprávy od exportéru na kolektor skrze nezávislé *proudy dat* (angl. *stream*). Sekvenční čísla přenášených zpráv jsou počítána pro každý proud dat v rámci transportního spojení nezávisle a tomu se i preprocesor musel přizpůsobit. U ostatních transportních protokolů (TCP a UDP), existuje pouze jeden proud. Lze tedy pozorovat, že pro plné určení původce se v datových zprávách zřetězené linky musí uvádět informace o transportním sezení, ODID a případná identifikace proudu dat v rámci sezení.

Preprocesor slouží také jako obraná linie, když přichází **chybně formátované zprávy**. Ať už se jedná o ledabyle implementovaný exportér nebo o úmyslně špatně formátovaná data, je nezbytné se s nimi řádně vypořádat. Problémem původního kolektoru bylo, že na tyto situace neuměl řádně reagovat. Za nejvíce problematické lze též označit, že nekontroloval, zda hranice sad nepřesahují hranice celkové zprávy nebo zda položka záznamu dynamické délky nepřesahuje meze příslušné datové sady. Takové počínání může vést až k havárii kolektoru. Nový preprocesor již musí brát na tyto situace důrazně ohled a detekovat je. Pokud tedy nově preprocesoru přijde chybně formátovaná zpráva, v případě spojení skrze TCP nebo SCTP pošle vstupnímu modulu přes speciální rozhraní požadavek na uzavření spojení s patřičným exportérem. Než se tak ale stane a vstupní instance potvrdí uzavření prostřednictvím zprávy sezení zřetězené linky, bude od tohoto zdroje všechny další příchozí zprávy, které se mohou nacházet v kruhovém bufferu, zahazovat. Opětovné vytvoření spojení je potom zodpovědností exportéru. Nepříjemnost zde ovšem opětovně přináší transportní protokol UDP, u něhož není ze strany kolektoru možné spojení uzavřít. Preprocesor se může bránit pouze zahazováním chybně formátovaných zpráv nebo rušením zpracovaných šablon, u kterých došlo k detekci špatně formátovaných záznamů. Pokud byl s konkrétní




Obrázek 5.13: Datová zpráva zřetězené linky v návaznosti na surovou zprávu

šablonou pouze dočasný problém, její příští periodická obnova zase umožní interpretovat její záznamy.

### Datový záznam a práce s ním

Zpracované datové zprávy se záznamy putují za preprocesorem dále skrze zřetězenou linku přes vnitřní až na výstupní moduly. Instance těchto modulů již mají všechny datové záznamy vyznačeny a nachystány pro pohodlnou práci. Nemusí si je v surových zprávách dohledávat a ani paranoidně ověřovat jejich korektnost. Přesto všechno se neobejdou bez další podpory ze strany kolektoru a jeho rozhraní. Nejlépe se způsob práce se záznamy demonstruje na příkladu použití.

Typický způsob práce s datovými záznamy je možné hledat u výstupních modulů. Nezřídka kdy musí převádět záznam z formátu IPFIX do odlišného formátu, který může být opět binární nebo i textový. Při konverzi se typicky prochází skrze všechny položky záznamu, vyčítají se jejich hodnoty a konvertují se do cílové podoby. Už zde lze spatřit tři důležité komponenty: šablony a jejich napojení na informační elementy, iterátory pro procházení skrze záznam a konverzní funkce. Samotné šablony a informační elementy již byly více než dostatečně probrány v dřívějších částech této kapitoly.

**Iterátor záznamu** slouží k pohodlnému procházení skrze veškeré jeho položky. Je až pozoruhodné, že u předchozí generace zcela chyběl. U statických záznamů (bez položek dynamické délky) by si možná každý modul vystačil i se samotnou zpracovanou šablonou, protože u každé položky je předem známá její pozice. Jakmile se ale v záznamu vyskytnou dynamické položky (např. řetězce), na pevnou pozici už není spoleh. Ten, kdo by záznam procházel, by si musel s každou zpracovanou položkou posouvat od začátku zprávy o její velikost, jenž se někdy vyskytuje v šabloně a jindy v datech samotných, což v praxi značně zvyšuje komplikovanost a hlavně chybovost. Jelikož procházení má význam u více modulů, celou situaci zachraňuje právě iterátor, který je schopen se potýkat s oběma případy a transparentně dynamické položky zpřístupnit. K celkovému pohodlí obvykle stačí, aby u každé položky zpřístupnil odkaz na její začátek, délku a návaznost na informační element zpracované šablony.

Samotná iterace však nemůže být jediným způsobem přístupu k položkám. Stejný význam má i náhodný přístup. Kolektor tak musí přirozeně nabízet i funkce k dohledání položky konkrétního informačního elementu. To může být například užitečné při filtraci záznamů, kdy se při vyhodnocení jednotlivých podmínek výrazu přistupuje na odlišné položky. Je zde vhodné poznamenat, že příznaky vedené u položek zpracované šablony mohou práci značně usnadnit. Například u nalezené položky je možné zjistit, zda se náhodou ještě nevyskytuje vícekrát či zda již není posledním výskytem.

Další významnou komponentou jsou **konvertory**. Protože veškeré datové záznamy jsou uchovány v původním formátování big-endian, je naprosto nezbytné, aby existovaly konverzní funkce, které umožní korektní práci s daty na běžných procesorových architekturách. To však není jediný důvod. Mnohem problematičtější je, že číselné hodnoty i stejného informačního elementu mohou být uloženy v různých délkách. Nový kolektor by tak měl poskytovat univerzální konverzní funkce, které je dovedou převést do stejné podoby. Například všechny bez znaménková čísla na odpovídající nejvyšší typ schopný pojmout všechny velikosti, tj. 64 bitovou variantu. Opačný směr, zápis, také nesmí být vynechán. Dosavadní zmíněné funkce konvertorů popisují pouze binární převody. Mnohdy se ukázalo, že stejně tak jsou významné konvertory z binárních hodnot na hodnoty textové. Ty naleznou uplatnění při převádění záznamů toků na textový formát (např. JSON), ale i při ladění.

**Biflow** patří mezi nově podporované druhy datových záznamů, a tak je nový návrh ani zde nemůže ignorovat. Velké břímě už na sebe přenesly šablony, které si pro každý směr pohledu vedou odlišné verze zpracovaných elementů a tím manipulaci značně usnadňují. Ať už jsou záměry modulu jakékoliv, může z příznaků šablony přímočaře určit, zda se jedná o šablonu obousměrného záznamu či nikoliv. Na základě toho je následně schopen se rozhodnout, zda na záznam bude nahlížet z odlišných směrů, což přijde vhod výstupním modulům při jeho rozkladu na dva jednosměrné toky, pokud výstupní formát biflow nebo jeho obdobu nepodporuje. Z pohledu práce se záznamy podpora biflow a různých úhlů pohledu by neměla chybět u iterátoru a ani při funkcích pro náhodný přístup.

# Kapitola 6

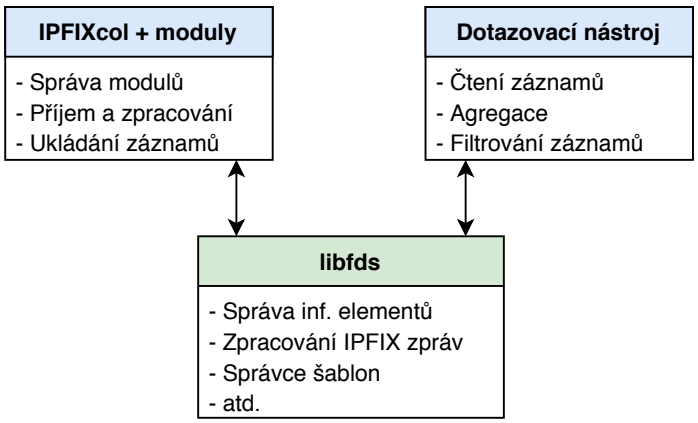
# Realizace kolektoru a měření dosažených výsledků

Nový návrh zřetelně ukázal, že změny se dotkly takříkajíc všech částí kolektoru. Ať už se jedná o správce šablon, který byl zcela od základů přepracován a je nově mnohem komplexnější, nebo přístup k práci se záznamy. Přidání nových funkcí nebo jejich úprava při zachování kompatibility se stávajícím rozhraním dosavadního kolektoru by byla velice nákladná, zdlouhavá a vnáší značné riziko vzniku chyb. Takové implementaci by ani nepomohlo to, že současné rozhraní kolektoru a funkce nejsou příliš vhodně dokumentovány a neexistují k nim komplexní automatizované testy. Jako logický krok se tedy jeví realizace zcela nové implementace, která je oproštěna od vazeb na staré rozhraní a do budoucna umožní další udržitelný rozvoj. Kvůli nesystémové implementaci stávajícího kolektoru a vzájemným závislostem nelze žádnou komponentu jeho jádra přímo převzít.

Dosavadní návrh uvedený v předchozí kapitole poskytl poměrně ucelený obraz o jednotlivých součástech, a proto je v této kapitole jádru kolektoru věnována pozornost pouze povrchově. Zejména jsou pojednávány detaily neuvedené v rámci nového návrhu, ale pro samotnou realizaci a další vývoj projektu významné. Samotné jádro je však zcela bezvýznamné, pokud pro něj neexistují zásuvné moduly. Pro tyto účely bylo vybráno několik dosavadních modulů ze stávajícího kolektoru a byly přizpůsobeny novému rozhraní. Přímo se tak nabízí unikátní příležitost srovnat výkonnost a posoudit výhody nových vylepšení. Závěr této kapitoly je věnován dalším návrhům a směřování projektu.

## 6.1 Struktura projektu

Oproštění od dosavadní implementace dává současně příležitost přehodnotit strukturu kolektoru v kontextu budoucího rozvoje a jeho nasazení jako prvku mnohem většího projektu. Z dříve uvedených vlastností víme, že kolektor umí data přijímat, zpracovat a dále je uložit nebo distribuovat. Jedním z palčivých problémů práce se záznamy toků je však schopnost později v nich dohledávat, co se na síti přihodilo. K tomu přirozeně napomáhají výstupní moduly, které konvertují záznamy do formátu, s nímž je možné dále pracovat a data procházet. Problémem je ale skutečnost, že současná řešení mají své limity. Buď není možné bezproblémově převést všechny vlastnosti toků, nebo je formát příliš těžkopádný. Například formát používaný populárním dotazovacím nástrojem *nfdump* nepodporuje položky dynamické délky, celé spektrum standardních položek z rozsahu IANA a už vůbec ne položky privátní. Do formátu JSON sice všechno zkonvertovat lze, ale formát je textový,




Obrázek 6.1: Postavení knihovny vůči kolektoru a dalším nástrojům

což je příjemné z hlediska přenositelnosti a zpracovatelnosti, ale vzhledem k nárokům na množství místa, které ve srovnání s binárním formátem potřebuje, formát není vhodný pro dlouhodobé uchování.

Řešení této situace se vcelku samo nabízí. Již samotný formát IPFIX je strukturován tak, aby ušetřil co možná nejvíce místa. Proč tedy nevzít data přijatá od exportéru a pouze je neuložit v relativně surovém stavu jako sekvenci na sebe navazujících přijatých IPFIX zpráv do souboru? Nejenže nebude třeba provádět žádnou výpočetně náročnou konverzi, nehrozí ani, že u záznamů dojde k oříznutí položek, u kterých není znám způsob konverze. Položky toků, které dříve kolektor nedokázal interpretovat budou stejně tak jako tak uloženy a když uživatel později dodá o jaká data se jedná (definuje odpovídající informační elementy), bude možné položky záznamů zpětně při dotazování interpretovat. Již tak úsporný formát jako jsou IPFIX zprávy lze navíc zavedením komprimace dat před uložením zmenšit, a snížit tak nároky na úložné místo. V mnohém se lze inspirovat v návrhu tzv. IPFIX File specifikovaného dle RFC 5655 [18]. Případný formát souboru je předmětem budoucího návrhu a je nad rámec této práce.

Z pohledu nástrojů, které by s takto vypadajícím formátem pracovaly, to ovšem znamená nutnost rozumět formátu IPFIX zpráv. Je přirozeně nevhodné, aby již tak komplikované součásti uvedené v návrhu musely být znovu samostatně implementovány. Lepším řešením je vytvořit sdílenou knihovnu, která by se přímo specializovala na práci s IPFIX formátem a také s budoucím souborem. Právě na tomto přístupu je postavena realizace nového kolektoru. Veškeré součásti potřebné pro zpracování zpráv jsou implementovány v nově vzniklé knihovně *libfds* (Flow Data Storage). Jmenovitě se jedná parser[1] IPFIX zpráv, správce informačních elementů, komponenty pro práci se zpracovanými šablonami, iterátory, datové konvertory a přístupové funkce záznamů. Vztah knihovny s kolektorem a případně dalšími nástroji demonstruje obrázek 6.1.

[1]Parser je komponenta provádějící rozbor IPFIX zprávy. Ověřuje správnou strukturovanost a určuje pozice šablon a jejich datových záznamů ve zprávě. Z pohledu kolektoru je důležitou součástí preprocesoru zpráv.

Nový kolektor a potažmo jeho zásuvné moduly jsou na samostatně distribuované knihovně závislé a notně ji využívají pro zpracování a přístup k záznamům. Samotné jádro kolektoru se tak může více soustředit na vytvoření prostředí pro běh modulů a jejich propojení. Knihovna bude mít v budoucnu ale dvojí využití, až vzniknou nebo budou upraveny nástroje, které budou analyzovat uchovaná data o tocích. V tomto ohledu lze zamýšlet například přizpůsobení dotazovacího nástroje *fdistdump*[2] nebo jeho derivátu nové knihovně a jejímu rozhraní. Vytvoření knihovny má zároveň pozitivum v tom, že zaručuje sdílení kódu a stejné rozhraní při zápisu i čtení, což v budoucnu dovolí přidávat další komponenty obecně užitečné při práci se záznamy.

Stejně jako předchozí generace kolektoru i její nástupce a nově vzniklá knihovna jsou z důvodu důrazu na výkonnost implementovány v programovacích jazycích C/C++. Zároveň platí, že všechna rozhraní mezi kolektorem, knihovnou a moduly jsou realizovány čistě skrze prostředky jazyka C. Původnímu kolektoru byla také vytýkána celá řada aspektů z oblasti způsobu vývoje. Ať už to bylo nesystematické pojmenování funkcí, zastaralý překladový systém nebo malý důraz na automatizovanou testovatelnost.

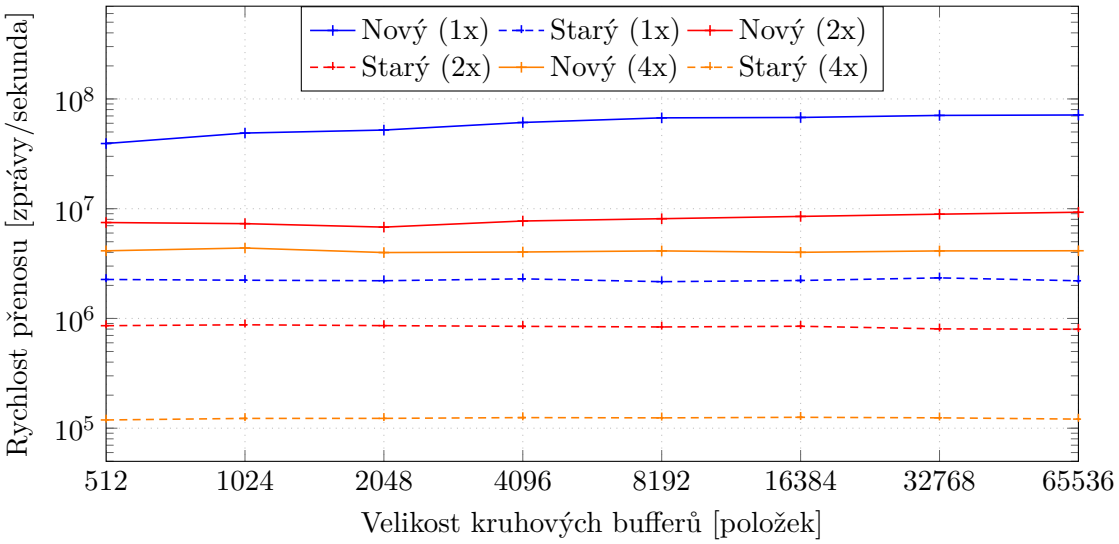
Dosavadní překladový systém *Autotools*, byl nahrazen za novější a stále více populární *CMake*[3], který také přímo poskytuje základní podpůrné prostředky pro automatizované testování. Zvolené nasazení *jednotkových testů* (angl. *unit testing*) přirozeně zvyšuje množství práce, které je při jejich tvorbě nutné vykonat. Snadno to ovšem vyváží reprodukovatelné ověření funkčnosti a korektnosti systému při jeho úpravách. Pro tvorbu testů byla zvolena osvědčená knihovna *Google Test*[4]. Protože ale není snadné určit, který kód je testy pokryt a který nikoliv, bylo v rámci implementace projektu do překladového systému doplněno automatizované generování pokrytí kódu nástrojem *lcov*[5]. Uvedený nástroj spustí testy nad speciálně přeloženou verzí projektu, posbírá statistické informace o jejich průbězích a výsledky prezentuje v podobě přehledné webové stránky.

Zásuvné moduly ve své implementaci při zpracování zpráv a jejich záznamů používají rozhraní kolektoru i zmíněné knihovny. Aby bylo jednoznačné, kde jsou funkce implementovány a komu náleží, používají se u všech identifikátorů prefixy `ipx` (jádro kolektoru) a `fds` (knihovna). Jako v dosavadním kolektoru se pro generování dokumentace ze zdrojových kódu používá běžně rozšířený nástroj Doxygen.

## 6.2 Jádro kolektoru

Z popisu rozdělení činností mezi specializovanou knihovnu a kolektor vyplývá, že jádro implementuje hlavně funkce starající se o moduly a jejich provázání. Unifikovaný kontext zastřešující každou instanci modulu a kruhové buffery, které je vzájemně propojují, představují asi jedny z nejvýznamnějších prvků. Nesmíme ovšem opomenout, že sem patří i preprocesor, výstupní manažer, různé druhy zpráv putující skrze zřetězenou linku a další. Co se týče kontextu, ten již byl dostatečně rozebrán v návrhové fázi. Jak se ale později v této kapitole ukáže, nový kruhový buffer nese nezanedbatelný vliv na celkový výkon kolektoru a je vhodné se jeho implementaci podrobněji věnovat. Za pozornost stojí i postup, jakým celé spouštění kolektoru probíhá.

[2] https://github.com/CESNET/fdistdump
[3] https://cmake.org
[4] https://github.com/google/googletest
[5] https://github.com/linux-test-project/lcov




Obrázek 6.2: Porovnání rychlosti kruhového bufferu starého a nového kolektoru při předávání ukazatelů na zprávy při využití jednoho čtenáře a rozdílného počtu zapisovatelů (uvedeno v závorce).

### Kruhový buffer

Při implementaci se mimochodem odhalilo, že dosavadní kruhový buffer pro předávání zpráv zřetězené linky nedosahuje dostatečné propustnosti. Pokud pomineme chaotické začlenění nevhodné a již dříve popsané postranní činnosti, dochází u něj k velice častému blokování vláken při přístupu ke společné řídící struktuře. Jinak řečeno, každý zápis a každé čtení znamenají přístup do sekce se vzájemným vyloučením, což má citelný dopad na jeho výkonnost. Buffer byl proto nahrazen za nový, který se snaží redukovat množství operací vyžadující zamykání. Stále se však jedná o princip podporující více současných zapisovatelů a pouze jednoho čtenáře.

Nová implementace obsahuje oddělené a patřičně zarovnané datové struktury pro vlákna provádějící zvlášť zápisové a zvlášť čtecí operace, aby se co nejvíce eliminovalo vyplavování záznamů z cache paměti procesorových jader. Celý buffer je navíc rozšířen o strukturu s výlučným přístupem představující synchronizační blok určený pro sdílení informací o pozici zápisové a čtecí hlavy. Jak struktura pro čtecí, tak i pro zapisující vlákna obsahuje jednou získané pozice hlav z opačné kategorie. Na základě toho uživatelé bufferu ví, zda mohou zapisovat nebo číst platná data, aniž by se vzájemně neustále synchronizovali. Pokud však narazí na situaci, že podle dříve zachycené pozice protistrany je buffer prázdný nebo naopak plný, přistoupí do společného synchronizačního bloku, odkud si vyčtou aktuální pozici protistrany, kterou si uchovají, a zároveň zapíší svoji. Podobný synchronizační postup se děje i v pravidelných intervalech, jakmile je vyčteno nebo zapsáno větší množství dat. V celkovém důsledku nedochází tak často ke vzájemným blokováním a výkonnost je řádově vyšší, což potvrzují i výsledky provedených testů shrnutých v grafu na obrázku 6.2.

Při testování[6] bylo vytvořeno vždy jedno vlákno čtenáře, které vyčítalo falešné ukazatele na zprávy řetězené linky generované patřičným množstvím vláken písařů, aby se co nejlépe nasimulovaly pozdější reálné situace. Současně byly vyzkoušeny různé kapacity kruhových

[6]Pro testování bylo použité stejné testovací prostředí jako je uvedeno v podkapitole 6.2

bufferů ve vztahu k maximální propustnosti. Z detailní analýzy se ukázalo, že při kapacitě 8192 záznamů u nového bufferů dosahuje průměrná přenosová rychlost více než dostatečné výsledky a další růst již není tak významný. Tato kapacita byla i z tohoto důvodu zvolena výchozí kapacitou kruhových bufferů používaných v kolektoru a následujících testech zásuvných modulů. Dále si lze z naměřených výsledků odnést poznatek, že současný zápis více vlákny způsobuje dramatické zpomalení. Tato skutečnost naštěstí tolik nevadí, neboť v novém kolektoru se může vyskytovat nejvýše jednou při spojování proudu zpráv od preprocesorů v případě více vstupních instancí.

### Konfigurace kolektoru

Doposud byly popsány všechny klíčové prvky kolektoru až na proces jeho spouštění. U běžných programů není nutné tuto problematiku příliš řešit, ale kolektor ze své podstaty představuje prostředníka pro zásuvné moduly, který je řídí a stará se o jejich spolupráci. Samotný start kolektoru se skládá z několika fází, kde může nastat libovolná chyba. Ať je to chyba při zpracování konfigurace, nebo selže inicializace instance zásuvného modulu, jádro kolektoru je schopno se s ní korektně vypořádat.

Po spuštění kolektoru a zpracování parametrů příkazové řádky se jako první vytváří správce informačních elementů a načítají jeho definice. Následně se začíná zpracovávat uživatelská konfigurace popisující instance modulů a jejich parametrů. Pro každou instanci se vytvoří zaobalující kontext a jádro se snaží v systému najít požadovaný zásuvný modul. Pokud je takový modul nalezen, dojde k ověření, zda je kompatibilní s aktuální verzí kolektoru a zda obsahuje všechny požadované funkce rozhraní. Teprve až jsou všechny kontexty připraveny, dojde mezi nimi k vytvoření kruhových bufferů a jejich provázání. V tu chvíli je možné přejít do fáze inicializace instancí. Nad každou instancí reprezentovanou kontextem se postupně volá inicializační funkce modulu a jsou jí předány uživatelem definované parametry na zpracování. Při inicializaci jádro postupuje v pořadí od konce, tj. od výstupních přes vnitřní až vstupní moduly. Pokud by se dle prvotního očekávání postupovalo od začátku ke konci, hrozí, že kdyby některé instanci vnitřního nebo výstupního modulu trvala inicializace nepřiměřeně dlouho, mohly by se systémové buffery na vstupu kolektoru zaplnit a už by docházelo ke ztrátě zpráv. Opačné pořadí inicializace tomu předchází. Až se podaří inicializovat všechny instance, každému jejich kontextu vytvoří samostatné řídící vlákno, čímž se zřetězená linka spustí. Nepodaří-li se nachystat ke startu byť jedinou instanci, všechny ostatní jsou ukončeny a s nimi i kolektor.

**Rekonfigurační proces** kolektoru tvoří jednu z jeho unikátních vlastností. Do současné implementace proces doposud nebyl zahrnut, jelikož není nezbytnou součástí a před nasazením vyžaduje pečlivé otestování. To nic nemění na faktu, že většina již hotových komponent s jeho podporou přímo počítá. Už teď je jasné, že budoucí implementace bude rekonfiguraci realizovat ve dvou fázích. V první dojde k porovnání původní a nové konfigurace, přípravě nových instancí a zároveň se skrze zřetězenou linku pošle konfigurační zprávu. Ta u instancí, jímž se změní provozní parametry, provede přípravu datové struktury aktualizace. Teprve až je vše připraveno, pošle se skrze zřetězenou linku další konfigurační zpráva, jenž aplikuje aktualizace, odebrané instance ukončí a nové vloží na správné místo v lince. Zde je důležité zdůraznit, že samotná linka se nikdy nezastaví, a to ani mezi zmíněnými dvěma konfiguračními zprávami.

| | 512 B | 1000 B | 1500 B | 2000 B | 2500 B | 3000 B | 5000 B | 10000 B |
|---|---|---|---|---|---|---|---|---|
| Běžná s. | 9 | 18 | 28 | 37 | 47 | 56 | 95 | 190 |
| Aplikační. s. | 4 | 9 | 14 | 18 | 23 | 28 | 48 | 97 |

Tabulka 6.1: Průměrný počet datových záznamů vyskytující se ve zprávě příslušné testovací sady (zaokrouhleno na celé jednotky)

| | |
|---|---|
| CPU | Intel Core i7-4765T (2.00GHz, TDP 35W) |
| RAM | 2x 8GB (1600MHz) |
| Operační systém | Fedora 27 |
| Nástroje pro překlad | gcc 7.3.1, cmake 3.11.0 |
| Parametry překladu | bez ladících symbolů, optimalizace O2 |

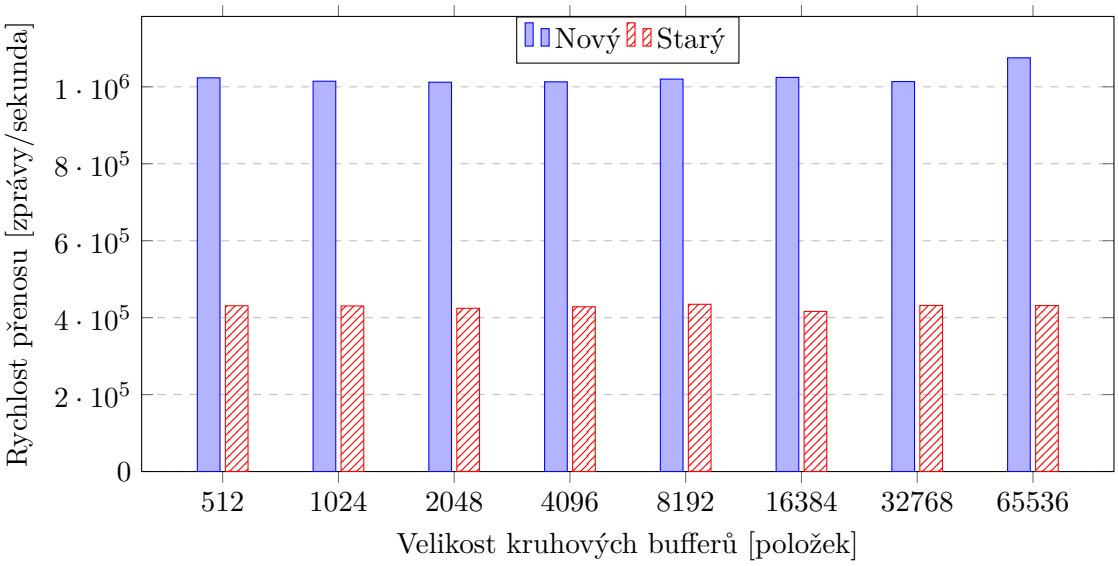
Tabulka 6.2: Parametry testovací sestavy

## 6.3 Zásuvné moduly

Jádro kolektoru a podružná knihovna pro práci s formátem IPFIX poskytuje nutný základ pro zpracování záznamů toků, ale veškerá další logika je součástí zásuvných modulů. Nezastupitelné postavení mají vstupní moduly pro příjem dat od exportéru. Stejně tak důležité jsou ovšem vnitřní a výstupní moduly, které záznamy toků dále zpracovávají, ukládají či distribuují. Pouze všechny součásti dohromady tvoří funkční celek. Vytvořit vybrané moduly pro nový kolektor bylo nezbytné. Část modulů bylo možné převzít z dosavadního kolektoru a pouze je upravit pro nové rozhraní a způsob práce s daty. Jiné moduly bylo nutné zcela nově od základu vytvořit.

Pro pozdější účely porovnání a testování zásuvných modulů byly přichystány testovací sady obsahující dříve zachycené reálné IPFIX zprávy. S cílem zjistit, jak se mění chování kolektoru v závislosti na velikosti a obsahu zpráv, byly vygenerovány dvě sady, kde jedna obsahuje záznamy se základními měřenými vlastnostmi a druhá měla toky rozšířené navíc o vlastnosti z aplikačních vrstev. Druhá sada, jinak řečeno, zahrnovala položky záznamů vyextrahované např. z DNS nebo HTTP provozu, a protože se četně vyskytovaly i textové řetězce, obsahovala sada oproti základní sadě záznamy variabilní délky. Pro každou z těchto dvou sad se dále vygenerovaly zprávy, jejichž maximální délka se lišila. V důsledku bylo tímto v testech možné nasimulovat, zda počet záznamů v jedné zprávě má vliv celkovou výkonnost kolektoru. V tabulce 6.1 se nalézá přehled použitých velikostí IPFIX zpráv a příslušný průměrný počet záznamů v každé z nich. Lze si všimnout, že množství aplikačních záznamů na zprávu je nižší než u záznamů běžných, jelikož každý záznam obsahuje více položek, a je tak přirozeně delší. Testovací sady byly na kolektor přehrávány za pomoci nástroje *ipfixsend2*, který je distribuovaný společně s novým kolektorem a podporuje transportní protokoly TCP a UDP.

Testovací prostředí, ve kterém byly všechny výsledky naměřeny, je uvedeno v tabulce 6.2. Jednalo se o fyzický stroj, na kterém se při testech nevyskytovaly jiné náročné úlohy.




Obrázek 6.3: Porovnání rychlosti zřetězené linky starého a nového kolektoru při přenosu prázdných IPFIX zpráv.

### Ukázkový vstupní a výstupní modul

Pro základní demonstraci a otestování rozhraní kolektoru byly vytvořeny velice primitivní ukázkové moduly. Hlavní činností vstupního modulu je produkovat IPFIX zprávy obsahující pouze povinnou hlavní hlavičku protokolu bez žádného dalšího obsahu. Tyto zprávy pouze zaobalí do datové zprávy zřetězené linky a pošle na zpracování preprocesoru. Pokud v konfiguraci instance není nastaveno vkládání umělého zpoždění mezi generovanými zprávami, produkuje jich co největší možné množství, dokud nedojde k jeho blokování z důvodu zahlcení kruhového bufferu při předávání zpráv. Výstupní varianta modulu naopak pouze přicházející datové zprávy z řetězené linky odebírá a vložením volitelného zpoždění tento proces může prodlužovat, aby simuloval pozdržení při běžném zpracování.

Moduly lze zároveň využít ke zjištění horní hranice propustnosti zřetězené linky. Pro tyto účely je vhodné využít konfiguraci obsahující právě jednu instanci vstupního ukázkového modulu a jednu výstupního. V okamžiku, kdy ani jeden z modulů do své činnosti nezanáší žádné umělé zpoždění, generuje se maximální možné množství zpráv, které projde skrze preprocesor a výstupní manažer až na výstup, kde dojde k jejich zpracování a zničení. Obrázek 6.3 ukazuje jaké je za této konfigurace maximální množství předávaných datových zpráv za sekundu. Pro tento účel bylo mimo jiné pro starý kolektor nutné vytvořit obdobný vstupní modul.

Z uvedeného obrázku je patrné, že nový kolektor je více než dvojnásobně rychlejší při přenosu zpráv. Pokud bychom vycházeli pouze z obecných poznatků srovnání rychlosti kruhových bufferů, pravděpodobně bychom mohli očekávat mnohem značnější rozdíl. Zde ovšem limitující faktor nepředstavuje zřetězená linka, ale rychlost alokace a uvolňování datových struktur IPFIX zprávy a zaobalující datové zprávy zřetězené linky. Jak je také vidno, ani nastavení velikost kruhových bufferů nemá téměř žádný dopad na celkovou přenosovou rychlost.

### Vstupní moduly pro TCP a UDP

Pro reálné nasazení kolektoru je klíčové umět přijímat data od exportérů. V tomto případě je nejvýznamnější podpora transportních protokolů TCP a UDP. Paradoxně standard protokolu IPFIX [4] vyžaduje, aby byl primárně implementována podpora pro transportní protokol SCTP, jehož podpora ale není doposud výchozí součástí většiny linuxových distribucí. Implementace tohoto patřičného modulu byla tudíž odložena na později. Oba implementované moduly nebylo možné ze staré generace kolektoru přenést, protože byly příliš svázány na původní rozhraní.

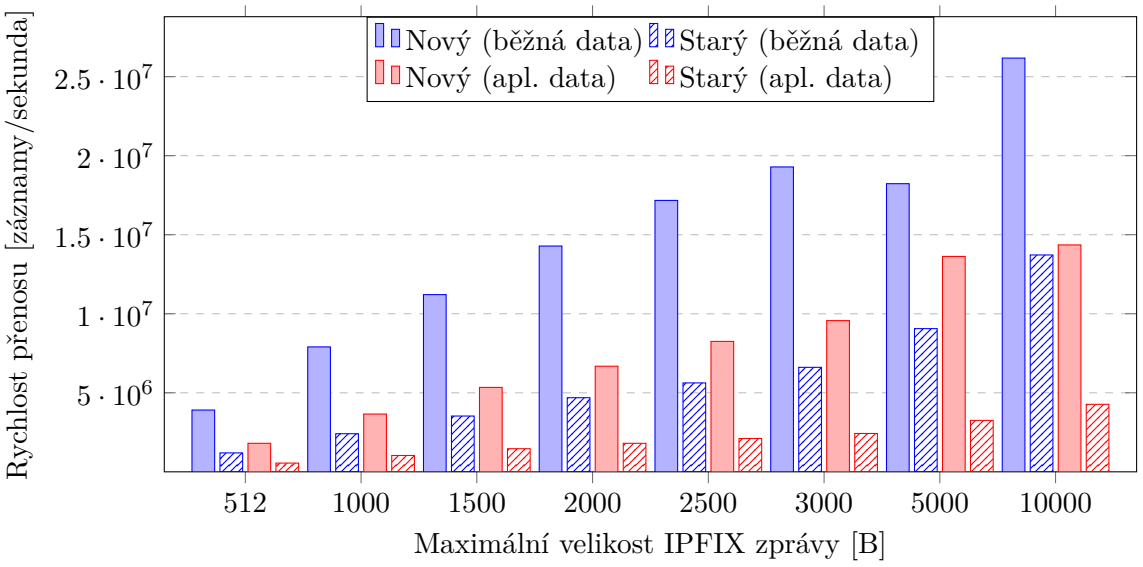
Vstupní **TCP modul** pracuje tak, že při svém startu vytvoří síťový soket a za jeho pomoci začne naslouchat nová příchozí spojení. Před zpracováním první datové zprávy zdroje je nejprve generována zpráva spojení signalizující ostatním instancím modulům, že došlo k vytvoření nového transportního spojení s exportérem. Data přicházející skrze transportní protokol TCP představují neohraničený proud dat a tento protokol ze své podstaty neumožňuje určit, jak velké datové zprávy byly protistranou spojení odeslány a kde jsou jejich hranice. Tuto roli zastupuje až samotný protokol IPFIX, který ve své hlavičce obsahuje celkovou velikost záznamu v bytech. Modul si tedy zpočátku vyčte hlavičku zprávy, provede ověření její validity, rezervuje dostatečně velké paměťové místo a ze soketu vyčte zbytek zprávy, kterou následně zabalí a pošle na zpracování preprocesoru. Poněkud komplikovanější je získání zpráv NetFlow, neboť v jejich hlavičce není přímo uvedena celková velikost, ale počet záznamů (NetFlow v5) nebo sad obsažených ve zprávě (NetFlow v9).

**Modul protokolu UDP** pracuje velice podobným způsobem, ale jisté odlišnosti zde lze přesto spatřit. Zprávy nepřichází v jednom proudu dat, ale jsou individuálně poslány v podobě samostatných datagramů, které si přímo nesou velikost přenášeného obsahu. Modul už nemusí pracně určovat z hlavičky, o jak velký záznam se jedná, ale předem si z rozhraní soketů vyčte jeho celkovou velikost a rezervuje pro něj dostatečné místo.

Společnou vlastností obou modulů je správa platných transportních sezení s exportéry. Součástí každé datové zprávy putující skrze zřetězenou linku je návaznost na identifikaci zdroje, od něhož zpráva přišla. Když se ale exportér odpojí, vstupní modul v reakci na tuto událost dá ostatním instancím kolektoru o situaci vědět prostřednictvím zprávy sezení. U protokolu UDP to ovšem není tak snadné, neboť zde se kvůli nespojované komunikaci kolektor nedozví, že druhá strana již žádná data generovat nebude. Dosavadní UDP modul tuto situaci nikterak neřeší. U nové implementace UDP modulu se z tohoto důvodu u každého záznamu sezení uchovává čas, kdy naposledy přišla poslední zpráva, a v pravidelných intervalech kontroluje, zda již nebyl překročen neaktivní časový limit.

Byť původní TCP modul správně informuje ostatní, že již došlo k ukončení spojení, oba původní moduly trpí stejný problémem. Datové struktury náležící dávno ukončených spojeních si uskladňují až do ukončení kolektoru, což potenciálně představuje bezpečnostní hrozbu v podobě vyčerpání systémových prostředků. Nové implementace využívají odpadních zpráv, které se o zničení zbytečných datových struktur postarají ve vhodný čas.

Pro účely srovnání výkonnosti vstupních modulů a jejích dřívějších protějšků byly vytvořeny konfigurace, kde se nacházela právě jedna příslušná instance vstupního modulu (TCP nebo UDP) a ukázkového výstupního modulu. Tento způsob dal příležitost ukázat, kolik záznamů je možné kolektorem přijmout za jednotku času. Na obrázcích 6.4 a 6.5 se nachází srovnání rychlosti příjmu TCP a UDP modulů pro starý a nový kolektor. Srovnání generací dává za pravdu, že oddělení vstupního modulu a preprocesoru zpráv v rámci nového návrhu bylo rozumné rozhodnutí. Nový kolektor zvládá přijmout znatelně více zpráv a v případě protokolu TCP je rychlost mnohdy více než dvakrát vyšší. Je nutno




Obrázek 6.4: Porovnání rychlosti přijmu dat vstupním TCP modulu starého a nového kolektoru nad běžnou a aplikační testovací sadou.

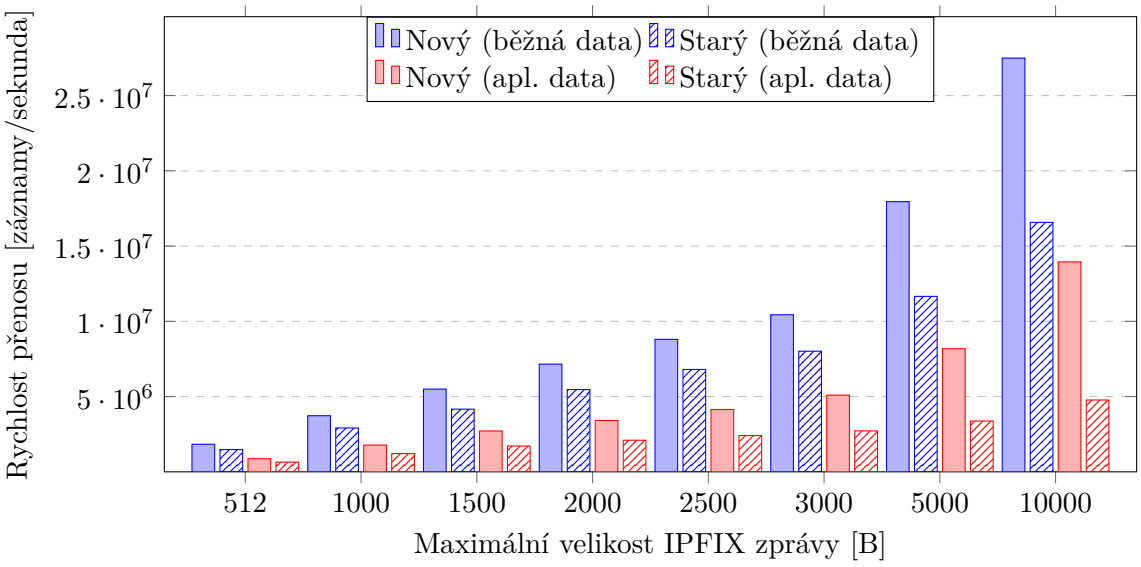
upozornit, že u protokolu UDP při vyšších velikostech IPFIX zpráv docházelo k zahazování malého poměru zpráv na straně systémových bufferů, neboť záznamy pravděpodobně nebyly dostatečně rychle odebírány. Odpovídající graf přesto ukazuje reálné množství záznamů, které skutečně přišly až na instanci výstupního modulu.

Výsledky množství přijatých záznamů jsou více než uspokojivé, ale přesto se nabízí otázka, zda by nešlo propustnost ještě zvýšit. Jednou z novinek nového kolektoru je i možnost paralelního běhu více samostatných instancí vstupních modulů, kde každá instance má vlastní preprocesor. Pro účely testování byly vytvořeny konfigurace s odlišným počtem instancí, kde každá naslouchala na vlastním lokálním portu a nástrojem *ipfixsend2* bylo na tyto port posíláno co nejvyšší množství záznamů. Výsledkem je srovnání propustnosti při různém počtu vstupních instancí prezentované obrázky D.1 a D.2, jenž jsou součástí přílohy D.

### Vnitřní anonymizační modul

Anonymizace toků je vhodnou ukázkou modifikace toků v jádru kolektoru. Identifikace koncových stanic, potažmo uživatelů, vychází zejména z IP adres přítomných v tocích a ně zrovna tento modul cílí. Pro anonymizaci je možné použít algoritmus Crypto-PAn [21], který mapuje bezkolizně jednu IP adresu na jinou a zároveň zachovává prefixové vlastnosti adres. Jelikož je založen na kryptografickém algoritmu, je velice výpočetně náročný, pokud nepoužívá specializované procesorové instrukce pro akceleraci výpočtu. Použitá implementace algoritmu bez podpory akcelerace dosahovala u obou kolektorů sotva rychlosti 100 000 záznamů toků za sekundu, a tudíž představovala úzké hrdlo ve zpracování toků.

Pro účely testování byla ovšem zvolena odlišná naivnější metoda, který zachová horní polovinu adresy a nahradí její spodní část nulovými hodnotami. Dojde sice k masivní kolizi adres, ale stále je částečně možné identifikovat zdrojovou síť. Původní verze modulu pokaždé vyhledávala nahodilým způsobem zdrojovou a cílovou IP adresu a provedla jejich úpravu. Ostatní položky informačních elementů, které by se také mohli v záznamech vyskytovat




Obrázek 6.5: Porovnání rychlosti přijmu dat vstupním UDP modulu starého a nového kolektoru nad běžnou a aplikační testovací sadou.

a obsahují IP adresu, jsou nejspíše z výkonnostních důvodů ignorovány. Tento modul byl pod nový kolektor převzat s tím rozdílem, že každý záznam se nově celý postupně prochází prostřednictvím iterátoru a u každé položky, jejíž typ informačního elementu je IP adresa, dojde k anonymizaci. To je také značný posun ve funkčnosti čerpající z nové podoby zpracovaných šablon a jejich provázání na informační elementy, neboť libovolná položka adresy může být na základě svého typu bez nutnosti zásahu do kódu modulu anonymizována.

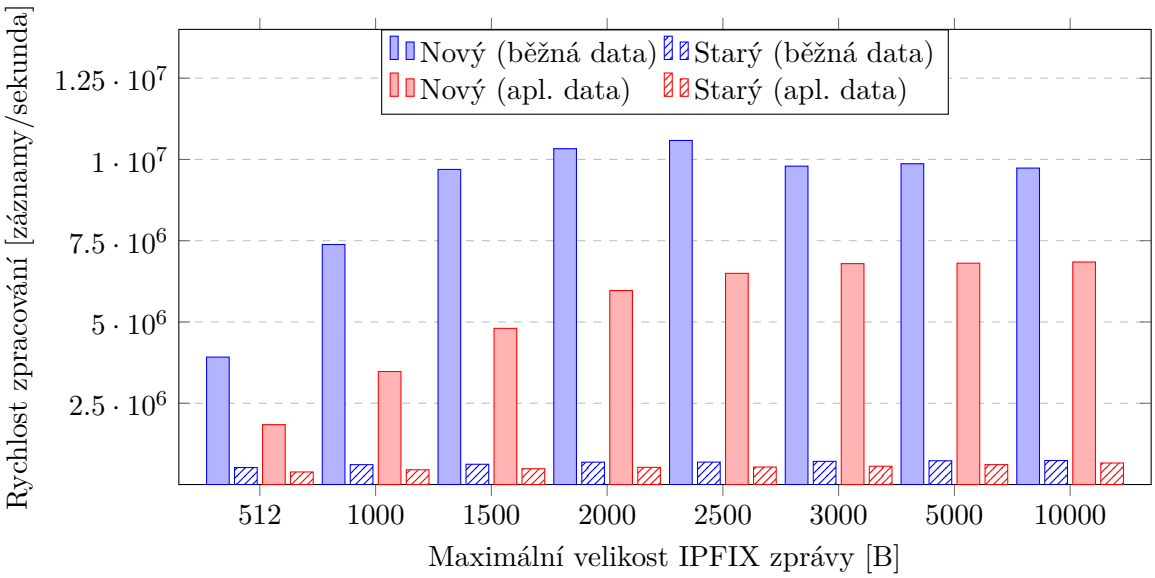
Obrázek 6.6 porovnává, s jakou rychlostí jsou záznamy modifikovány v rámci obou generací kolektoru. Jako vstup byl zvolen TCP modul a jako výstup posloužil jednoduchý ukázkový modul. Lze naprosto jasně pozorovat, jak nový kolektor a jeho funkce poskytují několikanásobně vyšší propustnost.

### Výstupní LNF modul

Jedním z příkladných zástupců výstupních modulů je modul konvertující IPFIX záznamy toků do binárního formátu používaného nástrojem *nfdump*, který je schopen později nad toky provádět mimo jiné filtrační a agregační činnosti používané při ruční analýze událostí v zachyceném provozu. Pro účely konverze využívá výstupní modul knihovnu *libnf*[7], která poskytuje rozhraní pro snadnou tvorbu souborů s konvertovanými záznamy. Nástroj *nfdump* a jeho formát značně vychází ještě z původního protokolu NetFlow a nelze tedy od něj očekávat, že by podporoval uchování veškerých položek toků. Jinými slovy při konverzi může docházet ke ztrátě méně obvyklých vlastností toku.

Celý proces převodu záznamu pracuje tak, že se postupně prochází jeho jednotlivé položky a v pomocné konverzní tabulce se hledá transformační funkce. Pokud je nalezena, je položka převedena. V rámci přizpůsobení novému rozhraní se zde využil zejména nový iterátor záznamů a konverzní funkce pro rozličné datové typy formátu IPFIX. Zpracování záznamu je tak o něco více bezpečné. Původní modul podporuje i ukládání do souborů založeném na tzv. profilování toků, o kterém se pojednává dále v navazující podkapitole.

[7] https://github.com/VUTBR/libnf




Obrázek 6.6: Porovnání rychlosti zpracování dat vnitřním anonymizačním modulem starého a nového kolektoru nad běžnou a aplikační testovací sadou.

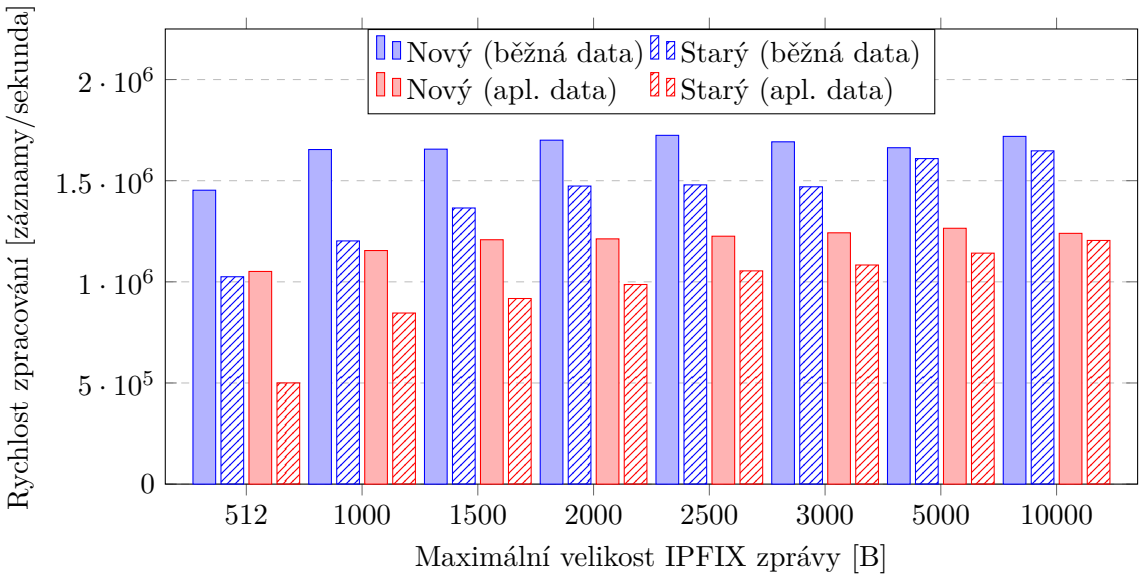
Podpora této již značně komplexnější činnost není v novém kolektoru integrována, a proto byla dočasně z převedeného modulu vyjmuta. Na druhou stranu, šablony nyní poskytují možnost rozlišovat biflow záznamy, a když se právě na takový záznam narazí, nový modul je schopen jej rozložit na dva samostatné jednosměrné toky a oba uložit.

Z obrázku 6.7 je patrné, že i zde došlo ke zvýšení množství převedených záznamů. Pro testování byla použita jedna výstupní instance v kombinaci se vstupním TCP modulem zajišťujícím zpracování dat. Veškeré záznamy byly ukládány s aktivní kompresí na disk. Stejně jako v dřívějších případech i zde je patrný výkonnostní nárůst, byť rozdíl není tak výrazný. Samotná transformace, komprese a zápis na disk mají nejspíše největší vliv na celkové zpoždění. Za povšimnutí stojí zejména fakt, že vstupní moduly zvládají do kolektoru přenést až řádově desítky milionů záznamů toku za sekundu, ale výstupní modul dokáže za stejné časové období zpracovat o řád nižší množství.

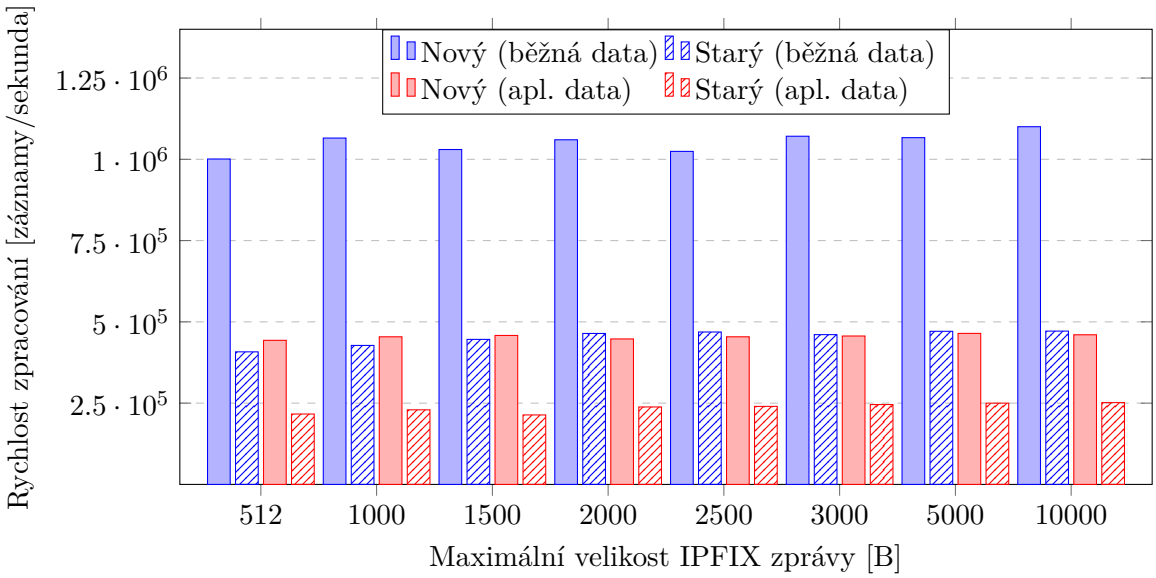
### Výstupní JSON modul

Posledním zástupcem z modulů pro nový kolektor je modul zajišťující převod do textového formátu JSON. Výhodou tohoto formátu je jeho snadná čitelnost a schopnost obsáhnout de facto všechny položky vyskytující se v záznamech toků. Proces konverze opětovně spočívá v postupném procházení skrze všechny položky záznamu toku za pomoci nového iterátoru a jejich převodu na textovou podobu. V modulu pro kolektor dřívější generace, kde šablony nebyly přímo provázány s definicemi informačních elementů, musel modul pokaždé dohledávat definici položky ve správci informačních elementů, aby věděl, jaké má textové označení a datový typ. Nový návrh kolektoru již tuto zdlouhavou činnost zcela eliminoval a také modulu umožnil použít řádně otestované konverzní funkce pro čtení a textový převod rozličných datových typů formátu IPFIX. Konvertované JSON záznamy je možné následně uložit do souboru, odeslat na vzdáleny server anebo je jako server nabízet klientům.

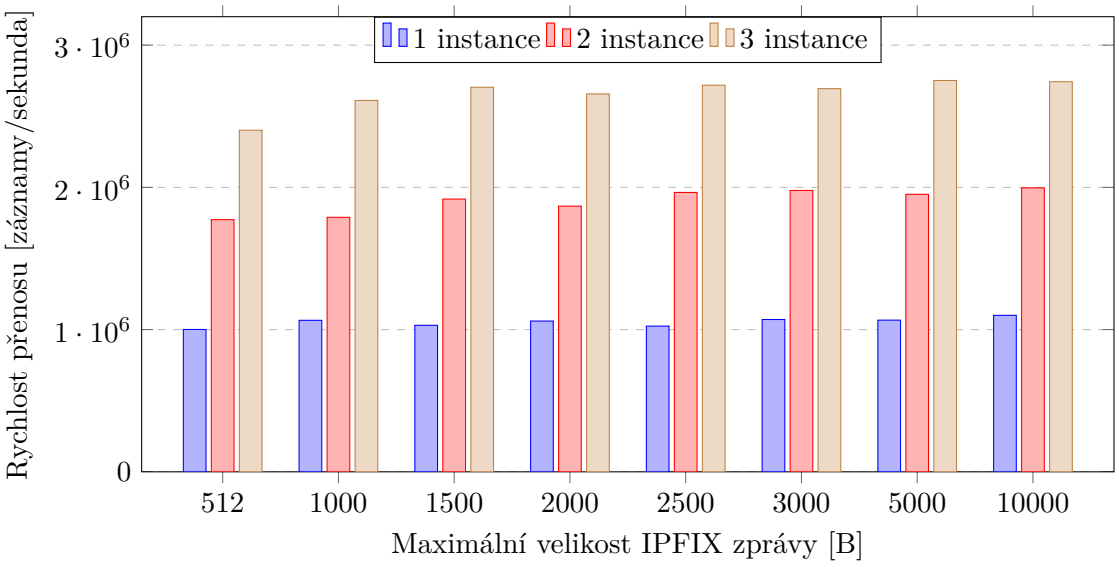
Obrázek 6.8 přináší srovnání obou generací tohoto modulu. Při testování byl použit jako vstup opět osvědčený TCP modul a zkonvertované JSON záznamy se na výstupu




Obrázek 6.7: Porovnání rychlosti zpracování dat výstupním LNF modulem starého a nového kolektoru nad běžnou a aplikační testovací sadou.




Obrázek 6.8: Porovnání rychlosti zpracování dat výstupním JSON modulem starého a nového kolektoru nad běžnou a aplikační testovací sadou.




Obrázek 6.9: Porovnání rychlosti zpracování dat výstupním JSON modulem nového kolektoru za použití více samostaných instancí nad běžnou testovací sadou.

zahazovaly, aby při měření výkonnosti nebyla instance modulu ovlivňována rychlostí odebírání záznamů jinou aplikací. Jak je možné z grafu vyčíst, nový návrh kolektoru přispěl k přibližně dvojnásobné rychlosti zpracování záznamů.

Obdobně jako u vstupních modulů se nabízí otázka, zda přeci jen není možné ještě rychlost zvýšit. Řešení se ze své podstaty neliší od přístupu použitého u vstupních modulů, neboť stejně jak může existovat více samostatných instancí vstupních modulů, může existovat i více instancí výstupních. Pro paralelizaci zpracování stačí, aby data přicházela s různými ODID a výstupní instance si rozdělily nepřekrývající se rozsahy, které zpracují. Při pokusu, jehož výsledek měření je možné pozorovat na obrázku 6.9, byla použita upravená konfigurace s patřičným množstvím výstupních instancí. Pro každou výstupní instanci byl kolektoru posílán separátní proud dat s odpovídajícím ODID. Z grafu lze vyčíst, že i tento přístup je poměrně dobře škálovatelný.

## 6.4 Budoucí rozvoj projektu

V rámci implementace se podařilo odstranit celou řadu nedostatků původní generace kolektoru. Nově navržený kolektor má současně do budoucna značnou příležitost pro svůj rozvoj, který by se měl zaměřit spíše na funkční stránku počínaje konverzí dalších zásuvných modulů. Jak bylo navíc na příkladech modulů demonstrováno, výkon nového kolektoru je více než povzbudivý a další zaměření na zvýšení výkonnosti není v tuto chvíli prioritní.

Aby mohl být kolektor použit pro široké nasazení, je vhodné se primárně zaměřit na komponenty, které doposud nebyly v rámci implementace integrovány. V tomto ohledu je asi nejvýznamnější doplnění konvertoru NetFlow zpráv na zprávy formátu IPFIX do preprocesoru. Zde se lze inspirovat v původním kolektoru, který potřebnou komponentu již integroval. Další významnou součástí je již několikrát zmiňovaný rekonfigurační proces, který zajistí, že i bez přerušení běhu kolektoru je možné nové moduly spouštět, stávající ukončovat nebo jim měnit běhovou konfiguraci.

Původní kolektor obsahoval i řadu rozšíření, která se věnovala profilování toků [8], což je možné popsat jako činnost, při které se tokům na bázi uživatelsky definovaných podmínek přiřadí příslušnost do jisté skupiny. Uživatel si může například rozlišit toky na základě vyskytujících se IP adres, komunikačního protokolu nebo i komplexních podmínek. Jednomu toku tak může být přiřazena náležitost ve více skupinách. Výstupní a vnitřní moduly, které rozumí profilovacímu označení, pak mohou na základě toho jednat a například všechny toky z jedné skupiny ukládat do společného umístění.

Pro účely profilování je nutné mít podporu pro filtr toků, který na základě pravidel uvedených v textovém výrazu dokáže rozhodnout, zda záznam toku podmínky splňuje či nikoliv. Jeden z těchto filtrů [16] již byl v původním kolektoru využit a jeho opětovné nasazení by si vyžádalo zejména jeho úpravu pro nové rozhraní. Tento filtr by také našel budoucí uplatnění při manipulaci s novým souborem pro uchování záznamů toků, o kterém se zmiňuje podkapitola 6.1.

Kolektor nemusí být pouze relativně jednoduchým středobodem měření toků, ale může být součástí většího celku. Dosavadní kolektor byl nasazen i v rámci distribuované architektury [20], kde více kolektorů spolupracovalo při příjmu a uskladnění dat. Z pohledu kolektorů se celá architektura zjednodušeně skládala ze 2 druhů samostatných kolektorů odlišných konfigurací, kde na čele stál hlavní kolektor, jenž distribuoval příchozí data na kolektory provádějící ukládání záznamů. Pro podporu tohoto schématu je nutné zejména přenést distribuční zásuvný modul do nového kolektoru.

# Kapitola 7

# Závěr

Úvodním tématem diplomové práce bylo nastudovaní monitorování síťových toků v počítačových sítích a využití aplikačního protokolu IPFIX pro přenos záznamů o naměřených tocích na kolektor. Na základě těchto znalostí byl následně analyzován dosavadní kolektor IPFIXcol, kde kromě rozboru samotné architektury byly prozkoumány i přínosy a úskalí existujícího řešení. Významná část této práce z provedeného rozboru vychází a představuje návrh kolektoru nové generace s celou řadou nových vylepšení a zásadními zásahy do architektury. Cílem práce bylo i nový návrh implementovat a přizpůsobit vybrané dosavadní zásuvné moduly novému rozhraní. Současná existence staré a nové generace poskytla významný prostor pro jejich srovnání z hlediska výkonnosti.

První část textu se zaměřila na určení postavení monitorování toků v oblastí sledování sítí. Následně byla pojednána celková koncepce síťových toků z pohledu standardů NetFlow a IPFIX. Text přibližuje komponenty a technologická řešení použitá od samotného zachycení provozu až po sběr záznamů o tocích a jejich analýzu, přičemž detailně poukazuje, jak jednotlivé fáze monitorování pracují s daty a navazují na sebe. Rozebrání protokolu IPFIX se zabývala další část. Popis protokolu se neomezil na pouhý výtažek ze standardů, ale cíleně se zaměřoval na souvislosti mezi jeho částmi, vlastnostmi a důsledky, které má jeho konstrukce na zpracování záznamů toků. Protože samotný protokol prošel vývojem, což se projevilo na řadě rozšíření, která se později objevila, byla ta s nejvýznamnějším dopadem na následný návrh také popsána.

Navazující část textu se věnovala rozboru dosavadní generace kolektoru IPFIXcol. Byly charakterizovány činnosti jednotlivých komponent, zprostředkován pohled na postupy zpracování dat a hlavně byla více přiblížena úskalí současného návrhu. Stav architektury se po mnohých stránkách ukázal jako nevhodný, což bylo zapříčiněno zejména nesystematickým návrhem a jeho chaotickým rozšiřováním o nové funkce. Nejen z těchto důvodu vznikl návrh zcela nové generace kolektoru, jehož primárním cílem bylo silné stránky původní architektury zachovat, a ještě více z nich vytěžit. Slabé stránky byly naopak nahrazeny novým řešením nebo vylepšením. Zcela zásadní přepracování postihlo všechny součásti od značného rozšíření správce informačních elementů přes šablony popisující strukturu záznamů toků až po konfiguraci kolektoru. Vznikly také nové komponenty a částečně se i změnil způsob pohledu na záznamy toků. Přetvoření se dočkala i vnitřní zřetězená linka, která v modulární architektuře hraje podstatnou roli a je nově více univerzální.

V rámci realizace došlo ke zcela nové implementaci kolektoru a byla souhrnně přehodnocena i celková koncepce z pohledu budoucího rozvoje. Jedním z faktorů, na který se kladl důraz, byl také budoucí rozvoj, což se projevilo tak, že část významných komponent pro

práci s formátem IPFIX byla oddělena do nově vzniklé samostatné knihovny, která umožní jejich využití i mimo kolektor.

Závěrečná část práce se věnovala konverzi a výkonnostnímu testování vybraných zásuvných modulů pod novým kolektorem a jeho rozhraním. Zde se teprve reálně projevilo, jaký má v této prací nově navržená architektura a implementace reálný dopad. Výsledky měření se ukázaly být více než povzbudivé, neboť ve všech sledovaných ohledech nový kolektor překonal svého předchůdce. V řadě případů dokonce vykázal několikanásobně vyšší výkon. Měření tak dalo za pravdu, že uskutečněné změny v nové generaci kolektoru míří správným směrem.

Touto diplomovou prací činnosti nekončí. Již v průběhu se mnohdy vyskytly směry, kudy se může kolektor dále ubírat. Fundamentálně je vhodné se zaměřit na rozšíření množství zásuvných modulů, což lze provést částečně konverzí modulů původního kolektoru. Přidání více možností, kde může být kolektor nasazen, také potenciálně umožní růst uživatelské základny. Další kroky by také měly mířit k rozvoji formátu pro dlouhodobé uchování zachycených dat v plné podobě, neboť se to ukazuje jako jeden z přetrvávajících problémů práce s toky.

# Literatura


[1] *NetFlow*. Wikipedia: the free encyclopedia, [Online; navštíveno 14.01.2018].
URL https://en.wikipedia.org/wiki/NetFlow

[2] Claise, B.; Bryant, S.; aj.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, RFC Editor, Leden 2008.
URL http://tools.ietf.org/html/rfc5101

[3] Claise, B.; Dhandapani, G.; Aitken, P.; aj.: Export of Structured Data in IP Flow Information Export (IPFIX). RFC 6313, RFC Editor, Červenec 2011.
URL http://tools.ietf.org/html/rfc6313

[4] Claise, B.; Trammell, B.; Aitken, P.; aj.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, RFC Editor, Září 2013.
URL http://tools.ietf.org/html/rfc7011

[5] Claise, B.; Trammell, B.; aj.: Information Model for IP Flow Information Export (IPFIX). RFC 7012, RFC Editor, Září 2013.
URL http://tools.ietf.org/html/rfc7012

[6] Hofstede, R.; Čeleda, P.; Trammell, B.; aj.: Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys Tutorials*, ročník 16, č. 4, Fourthquarter 2014: s. 2037–2064, ISSN 1553-877X, doi:10.1109/COMST.2014.2321898.

[7] Kekely, L.; Kučera, J.; Puš, V.; aj.: Software Defined Monitoring of Application Protocols. *IEEE Transactions on Computers*, ročník 65, č. 2, Únor 2016: s. 615–626, ISSN 0018-9340, doi:10.1109/TC.2015.2423668.

[8] Kozubík, M.: *Profilování dat pomocí IPFIX mediátoru*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2015.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=16962

[9] Leinen, S.: Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX). RFC 3955, RFC Editor, Říjen 2004.
URL http://tools.ietf.org/html/rfc3955

[10] Matoušek, P.: *Síťové služby a jejich architektura*. Publishing house of Brno University of Technology VUTIUM, 2014, ISBN 978-80-214-3766-1, 396 s.

[11] Mills, C.; Hirsh, D.; Ruth, G.: Internet Accounting: Background. RFC 1272, RFC Editor, Listopad 1991.
URL http://tools.ietf.org/html/rfc1272

[12] Mills, D.; Delaware, U.; aj.: Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, RFC Editor, Červen 2010.
URL http://tools.ietf.org/html/rfc5905

[13] Muenz, G.; Claise, B.; Aitken, P.: Configuration Data Model for the IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Protocols. RFC 6728, RFC Editor, Říjen 2012.
URL http://tools.ietf.org/html/rfc6728

[14] Quittek, J.; Zseby, T.; Claise, B.; aj.: Requirements for IP Flow Information Export (IPFIX). RFC 3917, RFC Editor, Říjen 2004.
URL http://tools.ietf.org/html/rfc3917

[15] Sivakumar, S.; Penno, R.: IP Flow Information Export (IPFIX) Information Elements for Logging NAT Events. RFC 8158, RFC Editor, Prosinec 2017.
URL http://tools.ietf.org/html/rfc8158

[16] Štoffa, I.: *Filtr IP toků*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=20008

[17] Trammell, B.; Boschi, E.: Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103, RFC Editor, Leden 2008.
URL http://tools.ietf.org/html/rfc5103

[18] Trammell, B.; Boschi, E.; aj.: Specification of the IP Flow Information Export (IPFIX) File Format. RFC 5655, RFC Editor, Říjen 2009.
URL http://tools.ietf.org/html/rfc5655

[19] Velan, P.: *Processing of a Flexible Network Traffic Flow Information*. Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2012.
URL http://is.muni.cz/th/255519/fi_m/

[20] Wrona, J.: *Optimalizace distribuovaného kolektoru síťových toků*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=18670

[21] Xu, J.; Fan, J.; Ammar, M. H.; aj.: Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, Listopad 2002, ISSN 1092-1648, s. 280–289, doi:10.1109/ICNP.2002.1181415.

[22] Čejka, T.; Bartoš, V.; Švepeš, M.; aj.: NEMEA: A framework for network traffic analysis. In *2016 12th International Conference on Network and Service Management (CNSM)*, Říjen 2016, s. 195–201.

# Příloha A

# Obsah přiloženého paměťového média

Kořenový adresář přiloženého média je strukturován následujícím způsobem:

| | |
|---|---|
| `kořenový adresář` | |
| `source/` | Zdrojové kódy aplikace včetně návodů pro překlad |
| `ipfixcol2/` | Nový IPFIX kolektor |
| `libfds/` | Knihovna libfds |
| `examples/` | Ukázkové konfigurace kolektoru |
| `doc/` | Zdrojové texty diplomové práce |
| `thesis.pdf` | Text diplomové práce |

# Příloha B

# Překlad a spuštění kolektoru

Pro překlad ze zdrojových kódů je nutné mít v systému nainstalované požadované závislosti:

- `gcc` (verze 4.9.0 a novější)
- `cmake` (verze 2.8.12 a novější)
- `libxml2` (včetně vývojářského rozhraní)
- `doxygen` (volitelně)
- `pkg-config` (volitelně)

Jelikož část funkcí kolektoru byla přenesena do knihovny libfds, je nutné ji přeložit před jeho překladem.

```
cd libfds
mkdir build && cd build
cmake ..
make && make install
```

Obdobně lze následně provést překlad samotného kolektoru nové generace:

```
cd ipfixcol2
mkdir build && cd build
cmake ..
make && make install
```

Výsledkem by měl být nainstalovaný kolektor, který je možné spustit s libovolnou konfigurací:

```
ipfixcol2 -c <konfigurace>
```

Nápovědu ke všem parametrům kolektoru je možné získat za pomoci přepínače `-h`.

# Příloha C

# Konfigurace nového kolektoru

Běhová konfigurace kolektoru dovoluje vytvoření instancí vstupních, vnitřních a výstupních modulů, které běží současně a jako celek spolupracují při příjmu a zpracování dat. Dokument konfigurace je zapsán pomocí značkovacího jazyka XML, kde na nejvyšší úrovní je konfigurace zastoupena kořenovým elementem `<ipfixcol2>`, který dále obsahuje elementy seznamu instancí vstupních, vnitřních a výstupních modulů.

| | |
|---|---|
| `<ipfixcol2>` | Reprezentuje kořen konfigurace pro kolektor nové generace. |
| ├─ `<inputPlugins>` | Seznam instancí vstupních modulů |
| │ └─ `<input>` | Konfigurace instance vstupního modulu |
| ├─ `<intermediatePlugins>` | Seznam instancí vnitřních modulů |
| │ └─ `<intermediate>` | Konfigurace instance vnitřního modulu |
| ├─ `<outputPlugins>` | Seznam instancí výstupních modulů |
| │ └─ `<output>` | Konfigurace instance výstupního modulu |

Obrázek C.1: Hierarchie XML konfigurace

Konfigurace instance modulu vždy obsahuje její označení pro případ rekonfigurace, název příslušného modul a seznam parametrů specifický pro konkrétní modul. Volitelně je možné předefinovat filtr úrovní výpisů od blokace veškerých oznámení přes chybové hlášky až po ladící výpisy. Bližší podrobnosti je možné najít v dokumentaci kolektoru. Označení a struktura je totožná u všechy typů instancí a pouze název zaobalujícího elementu se patřičně liší.

| | |
|---|---|
| `<input>` | |
| ├─ `<name>` | Označení instance |
| ├─ `<plugin>` | Identfikace zásuvného modulu v systému |
| ├─ `<verbosity>` | Filtr úrovně výpisů (nepovinné) |
| ├─ `<params>` | Seznam parametrů předaný instanci |

Obrázek C.2: Konfigurace instance vstupního modulu

Výstupní instance se odlišují navíc tím, že je možné přímo u nich specifikovat rozsahy ODID, které jimi budou zpracovány. Buď je možné specifikovat rozsah, který daná instance bude přijímat, nebo naopak rozsah, který bude ignorovat. Rozsah je čárkou oddělený seznam čísel a intervalů, např. “5-10, 12”. Pokud není uvedena žádná z variant, filtrace není aplikována a instance přijímá zprávy ze všech ODID.

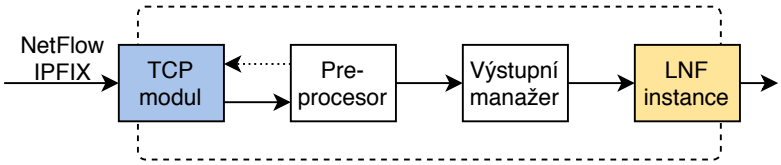
<output>

| | |
|---|---|
| <name> | Označení instance |
| <plugin> | Identfikace zásuvného modulu v systému |
| <verbosity> | Filtr úrovně výpisů (nepovinné) |
| <params> | Seznam parametrů předaný instanci |
| <odidOnly> | Filtr rozsahu ODID učených ke zpracování (nepovinné) |
| <odidExcept> | Filtr ignorovaného rozsahu ODID (nepovinné) |

Obrázek C.3: Konfigurace instance výstupního modulu

Nejlépe se konfigurace demonstruje na jednoduchém a zároveň poměrně typickém příkladu. Jelikož kolektor ke svému startu vyžaduje, aby byla definována instance alespoň jednoho vstupního a jednoho výstupního modulu, budou vybráni ti nejtypičtější zástupci z těchto kategorií. Jako vstup v našem případě poslouží instance TCP modulu, jenž, jak název napovídá, přijímá zprávy od exportéru skrze odpovídají transportní protokol, a výstup bude reprezentován instancí LNF modulu, který naopak veškeré záznamy ve zprávách zkonvertuje do formátu kompatibilního s nástrojen *nfdump* a data ukládá v podobě souborů pro dlouhodobé uskladnění. Protože v konfigurace se nenachází žádná instance vnitřního modulu může být element <intermediatePlugins> vyjadřující seznam instancí tohoto typu zcela nevychán.

Níže uvedený příklad výpisu XML konfigurace C.1 popisuje reálně spustitelný kolektor s uvedenými instancemi modulů. Na obrázku C.4 se pak nachází vyobrazení vnitřní architektury kolektoru při běhu této konfigurace. Vždy platí, že z pohledu vnitřního uspořádání kolektoru automaticky vzniknout vnitřní instance preprocesoru a výstupního manažeru.




Obrázek C.4: Struktura kolektoru při běhu z výpisu konfigurace C.1

Celkově se tedy uvedná konfigurace skládá ze dvou seznamů po jedné definici instance. V obou případech jsou libovolně vhodně zvoleny názvy instancí. Název zásuvného modulu pochopitelně musí odpovídat oficiální identifikaci a tedy “tcp” pro vstupní a “lnfstore” pro výstupní instanci. Parametry se modul od modulu liší. Úplný a podrobný výpis je obvykle součástí jejich dokumentace.

Vstupní TCP modul vyžaduje specifikaci, na jakých rozhraních specifikovaných IP adresou lokálního stroje bude naslouchat a jaký k tomu použije síťový port. Pokud není lokální adresa uvedena, jak je tomu v tomto případě, bude se naslouchat na všech. Na straně kolektoru je pro protokol IPFIX obvykle používán port 4739.

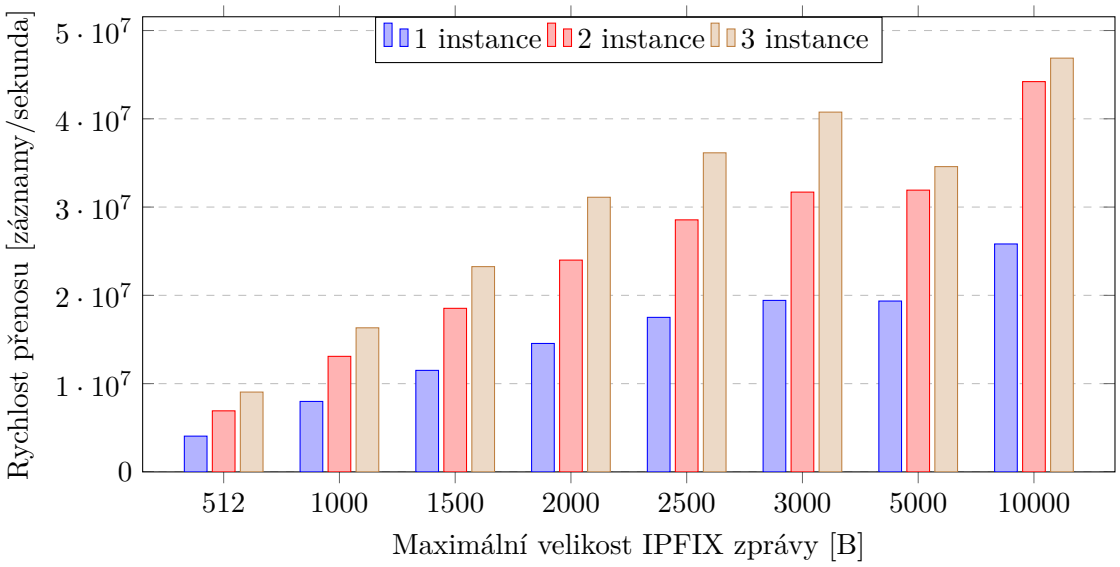
Výstupní LNF modulu ve své specifikaci minimálně požaduje určení cesty, kam budou soubory ukládány. Pokud není za pomoci dalších parametrů jinak určeno, vytváří v 5 minutových intervalech soubory z přijatých záznamů.

```
<?xml version="1.0"?>
<ipfixcol2>

  <inputPlugins>
    <input>
      <name>TCP collector</name>
      <plugin>tcp</plugin>
      <params>
        <localPort>4739</localPort>
        <localIPAddress></localIPAddress>
      </params>
    </input>
  </inputPlugins>

  <outputPlugins>
    <output>
      <name>LNF output</name>
      <plugin>lnfstore</plugin>
      <params>
        <storagePath>/tmp/flows/</storagePath>
      </params>
    </output>
  </outputPlugins>

</ipfixcol2>
```

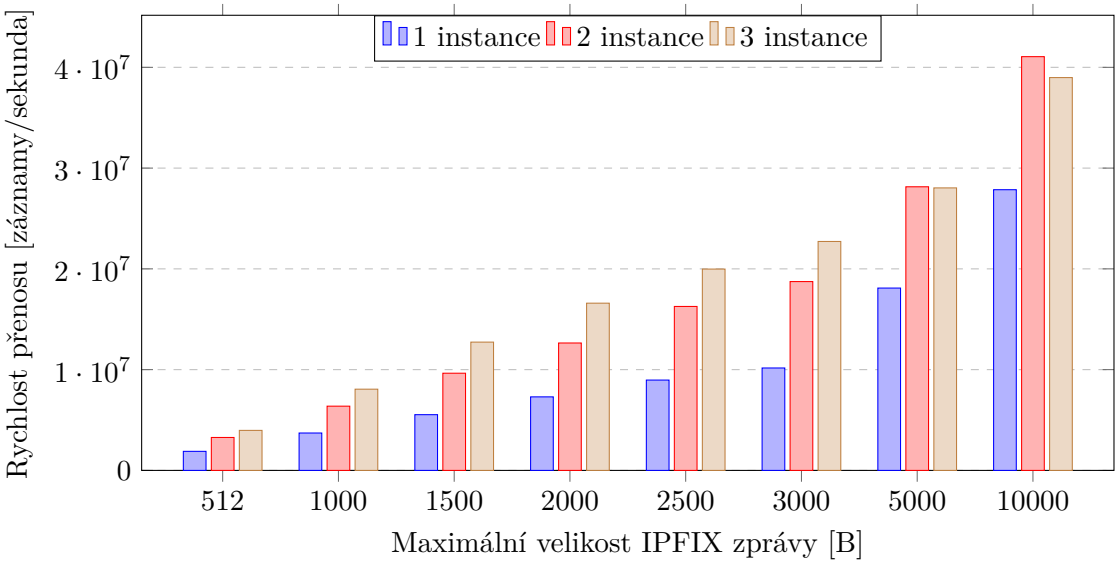
Výpis C.1: Konfigurace kolektoru s instancí jednoho vstupního a jednoho výstupního modulu.

# Příloha D

# Škálovatelnost instancí vstupních modulů




Obrázek D.1: Porovnání rychlosti přijmu dat vstupními TCP moduly za použití více samostaných instancí nad běžnou testovací sadou.




Obrázek D.2: Porovnání rychlosti přijmu dat vstupními UDP moduly za použití více samostaných instancí nad běžnou testovací sadou.